Authorship Attribution with Limited Text on Twitter

Luke Chen
Stanford University
Stanford, CA 94305
lukech18@stanford.edu

Eric Gonzalez
Stanford University
Stanford, CA 94305
ejgonz@stanford.edu

Coline Nantermoz
Stanford University
Stanford, CA 94305
coline@stanford.edu

ABSTRACT

We propose a collection of authorship attribution models that identify the author of Twitter messages. Natural language processing techniques extract lexical, syntactic, and semantic features, which are used as inputs to Naive Bayes, SVM, and neural network multi-class classifiers.

KEYWORDS

Authorship Attribution, Machine Learning, NLP, Twitter, Naive Bayes, Neural Networks, Sentiment Analysis, SVM, Word2Vec

1 INTRODUCTION

Authorship attribution has been a standard problem within Natural Language Processing (NLP). Stylometric analysis became renowned when it was used to attribute certain works to William Shakespeare and suggest potential collaboration on others, as well as to identify the likely writers of each of the previously unattributed articles in *The Federalist Papers*. Traditional author classification tends to use the entire corpus of the author's published work as training data. These works tend to be extensive, providing thousands of example sentences that, as part of a formal work, are usually similar in ideological content and structure.

The advent of the Internet and social media has brought a profound change in how it has allowed anyone, not just prominent figures, to express their ideas and create content for public consumption. The widespread use of social messaging platforms such as Facebook and Twitter, along with message boards and blog sites, means that there is an enormous amount of data available regarding how people write. However, the problem of author identification for the general public is made challenging by the differences between social media posts and traditional forms of writing such as books, newspaper articles, and research papers. One major difference is that for a typical person, the number of sentences available through social media posts is much smaller than what can be collected from an professional author's life output. Additionally, in contrast to traditional published works, social media posts tend to be more informal, concise (in some cases being subject to a word limit), and express a number of disparate ideas rather than support one coherent train of thought. The large corpus requirements of old authorship attribution models is a limitation that has prompted research into the shortcomings of traditional methods as well modifying techniques to adapt to the new trends in data.

As authorship analysis with social media posts is an area of active research and presents a number of useful applications, including personal identity verification and forensics, we have found the problem interesting and decided to investigate methods for authorship attribution on Twitter, a major source of public posts.

2 RELATED WORK

The classic model of authorship attribution by Mosteller and Wallace to classify *The Federalist Papers* used Bayesian statistics to distinguish the authorship of 12 disputed papers between Madison and Hamilton, using the remaining papers with known authors to train. In more recent years, attention has turned to applying these techniques to several authors at once using informal text that is limited in quantity. While Naive Bayes is still used as a standard benchmark with n-gram features, other supervised learning techniques have been used specifically to classify social media messages. Silva et al. [15] worked on authorship attribution for tweets in Portuguese, and obtained good results using SVMs and a small amount of data (from 100 to 2,000 tweets). However, they limited themselves to classifying among 3 authors. They showed that features such as emoticons, punctuation and emphatic expressions (such as "LOOOL") prevalent in informal messages are very informative in identification. Bhargava et al. [4] combined lexical, syntactical, tweet-specific, and metadata features with an SVM classifier and a RBF kernel to obtain high precision (up to 95%) with a a higher number of authors (10 to 20). Surprisingly, comparatively little recent work has been performed with neural networks on authorship attribution, an area we seek to explore.

3 DATASET AND FEATURES

3.1 Dataset

While our original goal was to use the Twitter API to scrape the tweets of users in real-time and generate this data ourselves, we realized this would be problematic. Twitter's search API restricts us from retrieving tweets from more than one week past, and it has a request rate limit, meaning we would only be able to collect a small random sample of tweets each day, making the volume of data required impossible to collect given the timeline. Without a publicly available dataset of tweets from prolific random users, we chose instead to use a pre-collected dataset of tweets from United States politicians [16], consisting of 1,243,370 tweets by 545 politicians (governors, legislators, executive office), with up to 3,200 of the most recent tweets by each user. It is true that politicians are not necessarily representative of the general public. However, we believe that the homogeneous and specialized nature of this group provides an upper bound in terms of classification difficulty and thus would still be a suitable dataset for our task.

3.2 Preprocessing

The initial dataset consisted of a single JSON file that contained the full metadata provided by the Twitter API (number of views, URL, user information, etc.) for each tweet. We manually isolated the tweets from the top 6 politicians with the most tweets (approx. 3,200). Due to memory limitations and time constraints, we limit the

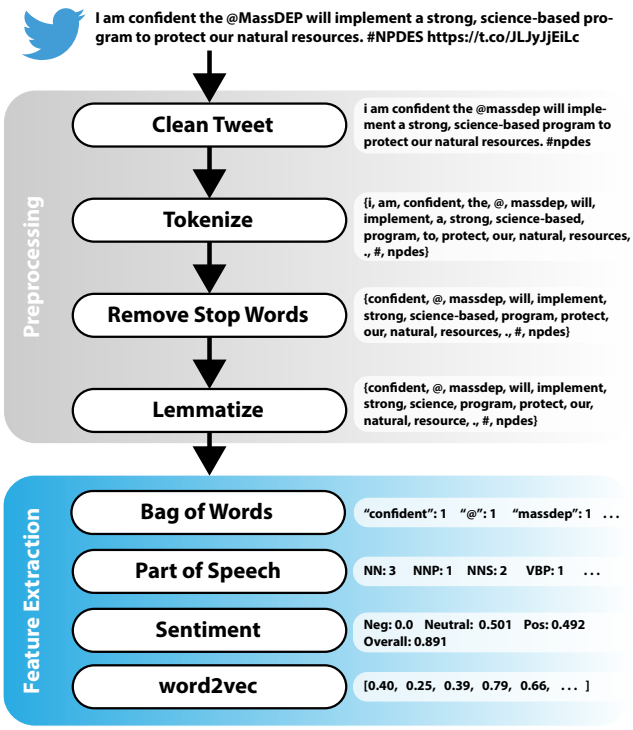

Figure 1: Data preprocessing and feature extraction pipeline with sample tweet and resulting features.

classification task in this work to at most 6 authors – however, the models and techniques discussed in the following sections would extend to an arbitrary number of authors.

We first filter out all data marked as a retweet, and "clean" the remaining tweets to remove links, numbers, and special characters (excluding traditional punctuation, #, and @, as we expected these characters to be relevant to author identification).

The cleaned tweets were then split into tokens using the NLTK package's *word_tokenize* function [10], which results in a list of words and punctuation. Very common words were removed by comparing against NLTK's built in *stopwords* set. A lemmatization algorithm was then used using NLTK's *WordNetLemmatizer* to reduce each token to its base lemma; for example: "runs", "running", and "ran" all reduce to "run".

3.3 Feature Selection and Extraction

Using the processed and tokenized tweet data, we extracted a set of lexical, syntactic, and semantic features for use in our classification models. The full data processing and feature extraction pipeline is shown in Fig. 1.

3.3.1 Bag of Words. We first use a "Bag of Words" (BOW) approach to count the unigram token frequency of each tweet. To generate this feature set, we first compile a complete vocabulary using every token occurring in the training set tweets, and assign each unique token a particular index. Thus if the training set contains 25,000 unique tokens, the unigram frequency feature of a single tweet consists of a 25,000 element array with the token frequency counts at their respective indices. Since the tokenization process separates a tweet into lemmatized words and punctuation, this feature encompasses the frequency of both. We chose to focus on unigram frequency over bigrams or trigrams, as previous research suggests unigrams are more effective for classifying short texts [8].

3.3.2 Part of Speech. To provide information beyond the lexical content of each tweet, we also introduce part of speech (POS) tagging to capture syntactical content. We use the Penn Treebank P.O.S. tagset [11] to assign each word (non-lemmatized) one of 36 potential part of speech tags, using the frequency of each tag within a tweet as features.

3.3.3 Sentiment. We conduct overall sentiment (SENT) intensity analysis on each tweet (nontokenized) according to VADER (Valence Aware Dictionary for Sentiment Reasoning), a lexicon-based sentiment analysis tool specifically for social media text[1]. VADER takes a whole tweet and outputs 3 numerical values (0 to 1) which correspond to the positive, negative, and neutral intensity of the tweet, as well as a single composite score, which ranges from -1 to 1. These four values are appended to the features above.

3.3.4 Word2vec. Since the use of unigrams results in a very large (although sparse) feature space, we investigated using word embeddings via Word2vec (W2V) [12] instead. Word2vec maps an entire text corpus into a vector space, typically several hundred dimensions large. Words that share common context in the corpus are closer together in the vector space, and thus in theory word embeddings capture both lexical and semantic information. We used a pretrained 200-dimensional Word2vec model trained on 2 billion tweets downloaded from [2]. To generate a single feature vector for an entire tweet, we follow the method described in [6] and maintain the max and min value of each dimension over all tokens in a tweet as a set of 400 features for that tweet.

4 MODELS AND METHODS

4.1 Naive Bayes

As a baseline for tweet classification, we implement a multinomial naive Bayes classifier with Laplace smoothing. By making the strong assumption of conditional independence between features, the likelihood of observing feature vector $\mathbf{x}$ given author A_k is

$$p(\mathbf{x}|A_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}^{x_i} \tag{1}$$

where p_{ki} is the probability of feature i occurring in a tweet by author k. By applying Bayes' Theorem and considering the log likelihood, we see multinomial naive Bayes can be expressed as a linear classifier:

$$\log p(A_k|\mathbf{x}) \propto \log\left(p(A_k) \prod_i p_{ki}^{x_i}\right) \tag{2}$$

$$= \log p(A_k) + \sum_i x_i \log\left(p_{ki}\right) \tag{3}$$

$$= b_k + \mathbf{w}_k^T \mathbf{x} \tag{4}$$

Given the assumption of discrete, positive elements of the feature vector, we explore only bag of words (i.e. word and punctuation

unigrams) and part of speech frequency features for classification in this model. We directly calculate the parameters p_{ki} and $p(A_k)$ using the distribution of training data features, with maximum likelihood estimation. For classifying the author of a given example, the log likelihood of each author k is calculated from Eq. (4) and the author with the largest likelihood is assigned as the class. We implement this model using scikit-learn [14] in Python.

4.2 SVM

We also explore two Support Vector Machine (SVM) models which perform multiclass classification using a one-versus-rest approach. In general, an SVM classifier for an n-dimensional feature space attempts to find the $(n-1)$-dimensional hyperplane that optimally separates the data; that is, the hyperplane which maximizes the "margin", or largest separation, between the classes. This is equivalent to solving the following optimization problem:

$$\max f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i K(\mathbf{x}_i, \mathbf{x}_j) y_j c_j \qquad (5)$$

$$\text{subject to} \sum_{i=1}^{n} c_i y_i = 0 \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda}$$

where λ is a regularization parameter, $y_i \in \{-1, 1\}$ is the label of example i, the c_i's are learned model parameters, and K is a user-selected kernel function defining a (potentially) higher dimensional inner product space. For a multiclass problem, the optimization is solved for all k classes, where $y_i = 1$ if example i is of the class of interest, and -1 otherwise. A new sample $\mathbf{z}$ is then classified by:

$$\text{class of } \mathbf{z} \equiv \arg\max_{j=1 \ldots k} \left(\sum_{i=1}^{n} y_i^j c_i^j K(\mathbf{x}_i, \mathbf{z}) - b^j \right) \qquad (6)$$

where b^j is an intercept solved for using a point on the margin boundary of hyperplane j.

In this work we explore SVM classifiers with a linear kernel (Eq. (7)) and radial basis function (Gaussian) kernel (Eq. (8)). We expect the linear kernel to perform better on sparse data such as word unigram features, and the nonlinear RBF kernel to perform better on the denser Word2vec data. We implement this model using scikit-learn [14] in Python.

$$K_{Linear}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z} \qquad (7)$$

$$K_{Gaussian}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right) \qquad (8)$$

4.3 Neural Network

We implement a neural network with Keras [5], using a TensorFlow backend [3], using grid search through scikit-learn [14] to tune hyperparameters.

For simplicity, we focus on a fully-connected feedforward architecture for the neural network. We have a linear stack of layers, where each unit in a layer is connected to all the neurons in the previous and following layers. The input layer matches the dimension of the feature set. The number of units in the output layer is the number of authors to choose from. The activation function of this layer is thus the softmax function, and the value produced by each output is the probability the tweet was composed by the corresponding author. We use a ReLU activation function units for all the hidden layers. In this work, we explore different optimization algorithms, network architectures, and regularization for a neural network tasked with text classification.

Keras allows the use of several gradient descent optimization algorithms. The most standard is Stochastic Gradient Descent (SGD), but it often yields poor results when data is sparse and heterogeneous (such as word frequencies). Therefore, an algorithm that adapts the learning rate for each parameter, performing larger updates for infrequent parameters for example would be more effective. Adam (Adaptive Moment Estimation) [9] for example makes use of both the first moment (corresponding to the average of recent magnitudes of the gradient) and second moment (square of each gradient term, corresponding to an uncentered variance). The process below describes Adam:

Let g_t the gradient of the objective function, at time step t.
At each time step we:
Update the biased 1st moment estimate: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
Update the biased 2nd moment estimate: $v_t = \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2$
Compute bias-corrected 1st moment estimate: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
Compute bias-corrected 2nd moment estimate: $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
Update parameters: $\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
β_1, β_2 and ϵ are parameters that generally take the values of 0.9, 0.999 and 10^{-8}.

Other optimization algorithms employ similar concepts: Adagrad [7] is similar to Adam but uses only the 2nd moment estimate. Adadelta [17] is a close variation, replacing the accumulation of past squared gradients with a decaying average of them. Adamax uses the infinite norm of the squared gradient instead of the L2 norm in calculating the 2nd moment estimate update. Nadam combines Adam with Nesterov accelerated gradient [13]. NAG is a way to "look ahead" and it computes a rough estimate of where the parameters are going to be in the next step, which helps to anticipate and prevent overshoot in one direction.

5 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we compare the models detailed in Section 4 in terms of classification accuracy on a dataset of 6 authors. We also analyze the impact of incrementally adding various features on the overall classification accuracy. Additionally, for the Neural Network model we also explore the impact of tuning various hyperparameters including optimization algorithm, network topology, and regularization.

In each of these experiments, our primary metric is classification accuracy as evaluated on a test set. Each model was trained on a set of 12,004 tweets and evaluated on set of 2,573 tweets.

5.1 Naive Bayes

As a baseline, we first evaluated the classification accuracy of a multinomial Naive Bayes classifier over multiple features. We also performed preliminary analysis on the effect of author quantity.

5.1.1 Features. Due to the strong assumptions required by naive Bayes, we evaluated the classifier using only Bag of Words (BOW)

		Naive Bayes		*Linear SVM*		*Gaussian SVM*		*Neural Network*	
Feature	Num Features	Train Acc. (%)	Test Acc.(%)	Train Acc. (%)	Test Acc.(%)	Train Acc. (%)	Test Acc.(%)	Train Acc. (%)	Test Acc.(%)
BOW	25187	95.1	83.4	100	85.0	–	–	100	**89.0**
BOW+POS	25231	93.2	**84.4**	100	87.6	–	–	–	–
BOW+POS+SENT	25235	–	–	100	**87.8**	–	–	100	82.1
W2V	400	–	–	71.9	68.4	99.8	**76.6**	92.0	68.4
W2V+POS	436	–	–	69.8	62.1	99.6	70.1	–	–
W2V+POS+SENT	440	–	–	665	60.4	99.6	71.0	91.4	66.5

Table 1: Accuracy results for classification of 6 authors using the proposed models and features. The best performance achieved by each model is in bold. The network shown is the best performing architecture tested: 1 hidden layer of 100 ReLU neurons.

and part of speech (POS) features. Since the model expects discrete positive features, we did not evaluate sentiment or Word2vec features with Naive Bayes. Table 1 shows the performance of the classifier with BOW features and BOW+POS features. On a pool of 6 authors, the Naive Bayes classifier resulted in a test accuracy of 0.84 with BOW features and did not change significantly with the addition of POS. This may be due to the vast difference in feature size between BOW (approx. 25,000) and POS (36).

5.1.2 Number of Authors. To gain insight into the effect of the number of authors on classifier performance, we trained and evaluated our Naive Bayes model on every combination of 2-6 authors in our dataset of 6 authors. For this task we used only BOW+POS features. Average test accuracy was 95%, 92%, 89%, 87%, and 84% for 2, 3, 4, 5, and 6 authors, respectively. As expected results show a reasonable drop in both both training and test accuracy with increase in the number of authors. This is one reason we were interested in examining other models, as Naive Bayes may not generalize as well to many authors.

5.2 SVM

The results of the two trained SVM classifiers are summarized in Table 1.

5.2.1 Linear Kernel. Training the SVM classifier with a linear kernel using identical features as Naive Bayes yielded improved test-set accuracy of about 2-3%. A regularization factor of 1.0 was empirically determined to maximize both training and test accuracy. We also see that adding syntactic and semantic features yielded improved performance for the Linear SVM model when using the highly sparse, linearly separable BOW-based features. On the contrary, Linear SVM performed poorly in all cases with the denser W2V based features.

5.2.2 Gaussian Kernel. We hypothesized that a Gaussian kernel SVM would outperform Linear SVM when training on W2V features since the reduced feature space is significantly more dense. Results indicate that this is true, with Gaussian SVM achieving approx. 8% higher accuracy on W2V features. A regularization factor of 10.0 was empirically determined to maximize both training and test accuracy. However, accuracy was still over 10% lower compared to Linear SVM's performance on BOW-based features. Note that due to the size and sparsity of BOW-based features, Gaussian SVM training did not converge in a reasonable amount of time and thus the model was not tested on these features.

5.3 Neural Network

In addition to the feature analysis similarly performed on the Naive Bayes and SVM models, we also investigated the tuning of various hyperparameters of the Neural Network, including optimization algorithm, network architecture, and regularization.

5.3.1 Features. We tested multiple sets of features for a neural network using a single hidden layer of 100 units, the Adam optimizer, a batch size of 100 and 30 epochs. These hyperparameters were selected based on the analysis in the following subsections. Surprisingly, the feature set containing only BOW yielded the best performance, as seen in Table 1, with 89% accuracy.

Our intent in including W2V features was to reduce the size and complexity of the feature space. While training time for 30 epochs went from 10 min to 7s, test set classification accuracy was greatly reduced to 66.5%. The addition of sentiment and part of speech slightly helped (68% accuracy) but remained far below the BOW-based features. The neural network experiments thereafter are done with the Bag of words feature set.

5.3.2 Optimization Algorithm. The optimization algorithms described in Section 4.3 were implemented with the default β / ϵ. The results are presented in Figure 2. As expected, the 6 algorithms that adapt the learning rate for each parameter perform better than simple Stochastic Gradient Descent. In particular, Adam, Adagrad, Adamax and Adadelta, due to their similarities, have comparable performances. We use the Adam optimizer for subsequent experiments.

5.3.3 Network Architecture. We investigated the number of hidden layers and units in each layer. We tested 1-3 hidden layers with various numbers of neurons but did not see significant impact on test accuracy, as shown in Table 2. We thus selected a single hidden layer of 100 neurons in order to improve speed.

5.3.4 Regularization. Our previous results indicate that our error resides in variance, not in bias, as our model overfits on the training set. Therefore, we explored adding regularization. As seen in Table 3, L1 regularization greatly reduces overfitting. However, it also lowers performance, dropping training accuracy to below 84%. Regardless of epoch size, the dev set never yields accuracy beyond 83%. L2 regularization and predominantly yielded comparable results to no regularization.

5.3.5 Final Tuned Model and Error Analysis. Having determined the optimal hyperparameters (one hidden layer with 100 ReLU activation units, using the Adam optimizer, with a batch size of

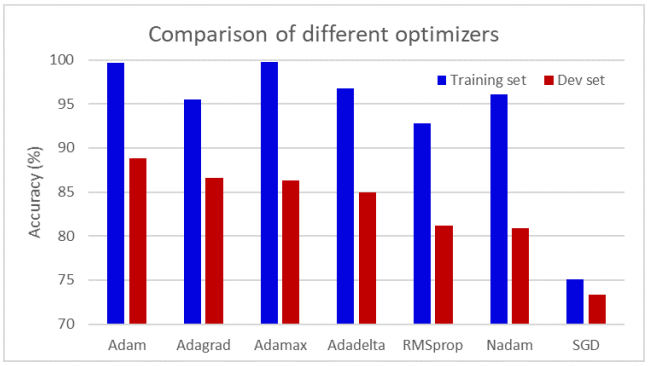

Figure 2: Comparison of the train and dev accuracy of several optimizers, all other hyperparameters of the neural network being equal (1 hidden layer with 100 neurons, batch size of 100 and 30 epochs).

Number of Hidden Layers	Size of Layer	Train Acc. (%)	Test Acc. (%)
1	50	100	88.5
1	100	100	88.5
2	100,50	100	88.2
3	100,60,20	100	88.1
3	200,120,40	100	88.3

Table 2: Comparison of the performance of different neural network architectures. We used Adam optimizer with a batch size of 100 and no regularization

Regularization (Parameter)	Train Acc. (%)	Test Acc. (%)
No	100	88.5
L2 (0.01)	92.7	83.6
L2 (0.001)	99.7	88.8
L1 (0.001)	83.5	80.0

Table 3: Influence of regularization on the performance after 30 epochs

100), we trained the neural network with all the optimal values together on bag of words features. The model achieves optimal test set accuracy after 7 training epochs; after this point, the model starts to overfit on the training data. As noted above, L2 regularization with small λ was not expected to give significant improvements and in creating the final model in fact decreased test accuracy; as such, the final model was taken without regularization.

Figure 3 presents the normalized confusion matrix after 30 epochs. The classifier often makes the mistake of attributing tweets to Author 4 (@onetoughnerd = Michigan Gov. Rick Snyder) when in reality they were written by Author 2 (@MassGovernor = Massachusetts Gov. Charlie Baker) or 3 (Kentucky Gov. Matt Bevin). Unfortunately, our data pipeline doesnt́ enable us to easily retrieve examples of misclassified tweets to analyze the reason. These 3

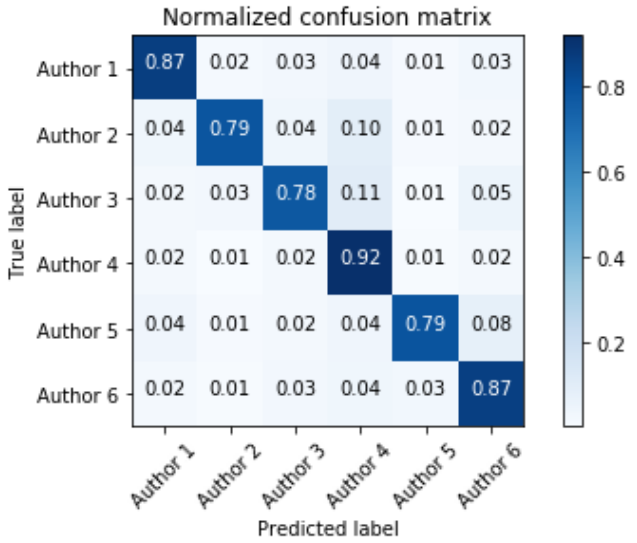

Figure 3: Normalized confusion matrix

authors are governors, whereas the 3 others are senators, and this common factor may be one reason for misclassification.

6 CONCLUSIONS AND FUTURE WORK

An author attribution model for limited text, evaluated on Twitter messages, achieves reasonable classification accuracy using a variety of Naive Bayes, SVM, and feedforward neural network approaches. Distinguishing tweets from 6 separate authors, nearly all model variants exceeded the performance of the baseline Naive Bayes model, which achieved a classification accuracy of 84%. Adding standard NLP features like part of speech and overall sentiment do improve accuracy for Naive Bayes and Linear SVM but not neural networks. The highest performing model was accomplished using a single ReLU hidden layer neural fully connected network with with only unigram features as inputs, and without performing regularization. This model achieved an 89% classification accuracy, indicating that the additional features in the other models, while helpful, were not as significant as changing the model. Additionally, since the baseline performance is already similarly high, we can conclude that although more powerful models do perform better, they do not overwhelmingly outperform. Word embeddings were investigated as an alternate method of including word-based features that would be advantageous due to lower feature dimensionality and captured context. However, on both SVM and neural network models, Word2vec features resulted in a much lower training accuracy even after some tuning, indicating the models are not powerful enough, leading to high bias.

There are a number of directions this work can be continued. We would like to determine what kind of neural network architecture would accommodate and reduce bias for Word2vec features, potentially adding more layers and investigating more complex structures like recurrent neural networks or LSTM. Additionally, we would like to further assess how our models perform as more authors are added and determine the factors that cause misclassification.

7 CONTRIBUTIONS

- Luke: Developed the overall architectural framework for generating files with feature matrices corresponding to parsed examples in Python. Implemented the data parsing, extraction from raw file format. Resolved technical implementation issues and contributed to grid search process. Provided the outline and wrote up the introduction, data and feature extraction, and conclusion for the poster and the introduction, related works, and conclusion for the final paper.
- Eric: Implemented feature extraction using NLTK, refined the SVM models, and ported Naive Bayes to Python. Worked on creating tables and graphs for the poster and wrote up the feature selection, methods, and results and analysis section in the final paper.
- Coline: Implemented the neural network using Keras and performed experiments to tune hyperparameters via grid search. Worked on creating tables and graphs for the poster and wrote up the related work, methods and results and analysis sections in the final paper.

REFERENCES

[1] Vader: A parsimonious rule-based model for sentiment analysis of social media text, author=Hutto, Clayton J and Gilbert, Eric, booktitle=Eighth international AAAI conference on weblogs and social media, pages=, year=2014, organization=.

[2] 2017. https://nlp.stanford.edu/data/glove.twitter.27B.zip. (2017).

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[4] Mudit Bhargava, Pulkit Mehndiratta, and Krishna Asawa. 2013. *Stylometric Analysis for Authorship Attribution on Twitter.* Springer International Publishing, Cham, 37–47. https://doi.org/10.1007/978-3-319-03689-2_3

[5] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras. (2015).

[6] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. 2016. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters* 80 (2016), 150–156.

[7] John Duchi, Elad Hazan, and Yoram Singer. 2010. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* Technical Report UCB/EECS-2010-24. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html

[8] Fatma Howedi and Masnizah Mohd. 2014. Text Classification for Authorship Attribution Using Naive Bayes Classifier with Limited Training Data. *Computer Engineering and Intelligent Systems* 5, 4 (2014), 48–56.

[9] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[10] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1 (ETMTNLP '02).* Association for Computational Linguistics, Stroudsburg, PA, USA, 63–70. https://doi.org/10.3115/1118108.1118117

[11] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics* 19, 2 (1993), 313–330.

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781 http://arxiv.org/abs/1301.3781

[13] Yurii Nesterov. A method of solving a convex programming problem with convergence rate O (1/k2).

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

[15] Rui Sousa Silva, Gustavo Laboreiro, Luís Sarmento, Tim Grant, Eugénio Oliveira, and Belinda Maia. 2011. *'twazn me!!! ;(' Automatic Authorship Analysis of Micro-Blogging Messages.* Springer Berlin Heidelberg, Berlin, Heidelberg, 161–168. https://doi.org/10.1007/978-3-642-22327-3_16

[16] uStuckInTheMatrix. 2017. Over one million tweets collected from US Politicians (President, Congress and Governors). https://www.reddit.com/r/datasets/comments/6fniik/over_one_million_tweets_collected_from_us/. (2017).

[17] Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR* abs/1212.5701 (2012). http://dblp.uni-trier.de/db/journals/corr/corr1212.html#abs-1212-5701