Who Let the Tweets Out?

CS6140 Project Report

Team Name: Delphi's Disciples (Aviral Goel and Akshar Varma)

Abstract

We study the problem of author attribution for very short texts, focusing on the task of feature engineering/extraction. While there are many studies that look into how to perform feature engineering for the author attribution task, all of those focus on long essays or novels unlike our focus on tweet length texts. With such short texts many of the features normally used become statistically insignificant and we need other ways of feature extraction from the texts. The primary focus of our project is on comparing explicit linguistic features like Bag of Word ngrams of the text against latent space embeddings [1, 2] that aim to capture semantics of the texts.

1 Introduction

Author attribution/identification is a well studied problem where given a text, the goal is to identify the author of the text. A common approach to studying the problem is to focus on the features extracted/engineered from the text that are most useful in identifying the original authors. Whereas generally, research has primarily focused on long texts like novels or essays, we look at the more niche problem of solving the task given very short texts [11, 15, 21]. This is an important subproblem because in scenarios like email forensics [7] we may not have a corpus with large texts. In our project we focus on very small text lengths by using a corpus of tweets of US politicians [3] as detailed in Section 4.

While we only have access to very short texts (each limited to 140 characters), we have access to many such texts (hundreds per author). This is in contrast to a lot of the author attribution literature which deals with essay/novel length texts (hundreds to thousands of words) but only 10s of texts per author. On the other hand, like many other works in the literature, we also focus on comparing and understanding which features are best for the author attribution task. We provide a more detailed comparison to related author attribution works as well as the choice of the sentence embedding technique we use in Section 2.

The main challenge in our problem is that the success [22] of the author attribution problem given long texts cannot directly transfer to short texts. With long texts there is enough stylistic information about the author that can be inferred. Some authors prefer to use long sentences, while others prefer shorter sentences. Or perhaps the difference is in how frequently big words are used, or how punctuation is used. However these kind of statistics are not significant if we only have short texts during training/prediction. This make the task much harder since most of the linguistic features that are normally used become ineffective. Thus it becomes crucial to understand what features are the most useful for short text author attribution.

In Section 3, Section 4.3 and Section 5 we discuss our methodology, experiments and analysis; elaborating on the following contributions of our project:

- Confirm empirically that linguistic features perform poorly on short texts.
- Provide conceptual justification for using Bag of Word ngrams and sentence embeddings as features for short texts.

Preprint. Work in progress.

- Scale the Bag of Words representation to thousands of unique ngram tokens using a hashing based compression scheme. This is crucial when there are thousands of texts per author and 10s of authors as is the case with tweets.
- Evaluate empirically the prediction accuracy of the Bag of Word ngrams features and sentence embedding features using various standard machine learning methods to find the better set of features.

2 Related Work

The problem of author attribution was first studied in 1964 by Mosteller and Wallace to settle the dispute of the authorship of the Federalist papers [14]. Since then, most methods for author attribution have used explicit features [20] from the text (eg. word/sentence lengths, comma counts, verb/adjective information etc.) on which statistical analysis is performed. Another common approach to feature engineering is to represent the text as a bag of words and feed that as input to learning algorithms like SVMs or Neural Networks [4, 5, 7, 9]. These methods have found success in recent years, for example, they were used to determine that the author of the book *The Cuckoo's Calling* was J. K. Rowling who was writing under the pseudonym of Robert Galbraith [22].

Whereas these works primarily focus on long texts like novels, we look at the more niche problem of solving the task given short texts [11, 15, 21]. An important motivation for this subproblem is in scenarios like email forensics [7] where we don't have a corpus with large texts. Another motivation is detecting twitter bots and recognizing propagation of fake news by identifying the original authors/bots of tweets. Our focus on tweets is similar to the Stanford ML course project by Chen, Gonzalez and Nantermoz [6]. In contrast, while their focus was on coming up with and comparing various learning models, we focus on comparing features using standard learning models.

The classical methods for author attribution primarily use lexical and syntactic features of the text which are often extracted using statistical methods. However, for small texts like tweets these methods are statistically insignificant and perform poorly. Hence, we use the idea of latent space embeddings of the texts that capture semantic/stylistic aspects as our features. Latent space embeddings were first used for words in works like word2vec [12] and GloVe [18]. Word embeddings have shown promising abilities [13] in capturing semantics and there has since been a lot of work on extending these embeddings to get sentence embeddings [8, 10, 16]. To keep our computational requirements low, we look at the work of Arora et al. [2] which provides a simple framework that gives a strong baseline that is reasonably competitive against more complex methods like ELMo [19]. Using these sentence embedding methods provides an opportunity to understand how well author specific characteristics are captured by these semantics focused methods.

3 Methodology

At a high level our methodology revolves around using multiple feature extraction/engineering approaches fed into various standard ML algorithms to compare classification accuracy. However, instead of focusing on which algorithms lead to good accuracy, we focus on determining which features work better. The features we want to compare are explicit linguistic features, Bag of Word representations and latent space embeddings. The motivation for comparing these features is to understand their relative capabilities and their contrasting strengths and weaknesses, particularly in the context of short texts. In the following sections we explain each of these feature and explain why they are important to compare.

3.1 Explicit Linguistic Features

The standard methods which are based on lexical and syntactic features of the text do not work directly on any representation of the text, but on statistical features extracted from the text. As detailed in Section 2, there is a lot of literature on what kind of features are important/discriminative and which feature is the most useful in different contexts. We use all the major features shown to have good performance that make sense in the context of very short tweets and pass them all on to the learning models.

The features used are:

- Average words per sentence
- Average characters per word
- Number of punctuations
- Number of words
- Number of characters

The reasoning behind the above features is that some people tend to write longer sentences, or use bigger words, while others may use more commas or other punctuation. These features capture such syntactic and lexical choices that authors make. We avoid using some features which are often used in literature since they don't make much sense in very short texts. For example, rare words are not prevalent enough in tweets (most have 0) for any learning model to infer anything from that information while adding another dimension to the input data.

Explicit linguistic features are standard features for the author attribution task but they are less powerful when the amount of data available is limited and their statistical significance itself becomes questionable with very short texts. We hypothesize that with very short texts, these features do not perform well and show supporting empirical evidence in Section 5.

3.2 Bag of Word ngrams

The Bag of Word ngrams representation uses a vector of length equal to the number of unique ngram tokens in the corpus and puts the frequency of token i in index i.[1] Thus the Bag of Word representation captures all of the text and hence doesn't face the statistical insignificance issue faced by the explicit linguistic features.

These features have also been known to work well for author attribution and have been used in many of the related works just like the explicit linguistic features (refer Section 2). However, such a representation is very memory-inefficient since most short texts will have a sparse bag of words representation. We overcome this using a hashing technique described in the next section.

3.2.1 Hashing to compress sparse vectors

We reduce the memory requirement by hashing down the vectors to be of 300 dimensions instead of number of unique tokens (which can be of the order of 10^5). This is done by taking some hash function and instead of putting the frequency of token i in index i, we put it in $hash(i)\%300$.

Hashing down means that we lose information due to collisions but this is not significant since for short texts the total number of tokens in the text will be quite low (low 100s for tweets) and hence collisions of tokens will be low[2]. Consequently, the final hashed down vector will be unique with high probability while becoming scalable in terms of memory. For example, we set n to be 2 and 3, to use Bag of Word bigrams and Bag of Word trigrams in our final experiments. For our dataset (described in Section 4) hashing reduced the size of a Bag of Words vector from 4569010 (for bigrams) and 11700934 (for trigrams) to 300.

3.3 Latent Space Sentence Embeddings

These are important to consider since sentence embedding methods have been shown to be able to capture semantic characteristics of sentences (tweets in our case), something neither the explicit linguistic features nor the Bag of Word ngrams can do. However it is unclear whether the semantic characteristics that are captured also capture stylistic characteristics well enough to distinguish authors. We use the embeddings described in the next section as our features.

3.3.1 Smoothed Inverse Frequency Embeddings

The embedding method we use is the Smoothed Inverse Frequency (SIF) embedding technique by Arora et al. [2] which starts with embeddings for all the words in a sentence and takes a weighted

[1] Alternatively, a simpler approach would be to use a 1/0 boolean in index i to represent whether token i appears in the text or not.

[2] Information theoretically it can be argued that with the right hash function we will only need a few 100 slots and it can also be argued that a random hash function will not behave too differently.

average of those embeddings to get the embedding of the sentence. The embedding v_S of a sentence S can be described mathematically as follows, where w is a word in the sentence, v_w is the embedding vector of the word w, and f_w is the frequency of the word w in the corpus.

$$v_S = \sum_{w \in S} \frac{a}{a + f_w} \cdot v_w$$

Here the weight term $\frac{a}{a+f_w}$ gives less weight to more frequent words thus implicitly capturing rare words and meanings in the sentence embedding. The hyperparameter a is meant to smooth the weight term and we set it to be 10^{-3} according to the setting from the paper on SIF embeddings. We start with pretrained GloVe embeddings of 300 dimensions to get our v_w vectors resulting in a final sentence embedding dimension of 300.[3]

A secondary step that can be performed on the SIF embeddings is to remove the principle component from v_S, where the principle component is of the embeddings of a small sample of sentences. This is done because the principle component is related to grammar and stop words and it has been shown that removing this allows the embeddings to better capture sentence meanings. We use both variants in our experiments.

3.4 Learning models used

Each of the above features is used as input to the following standard machine learning classification models using standard parameter settings from the scikit-learn Python library [17].

1. Random Forest: 10 estimators
2. Gaussian Naive Bayes
3. SVM: Squared Hinge Loss, L2 Penalty
4. Gradient Boosting: Deviance Loss, 100 Estimators
5. AdaBoost: 50 Unit Depth Decision Tree Estimators, SAMME.R Boosting Algorithm
6. ExtraTrees: 10 Randomized Decision Trees with Gini impurity criterion
7. Bagging: 10 Decision Trees
8. Neural Network: 1 hidden layer, 100 neurons with ReLU activations

We also use the following distance metrics in a k-nearest neighbor classifier

1. Bray-Curtis: $dist(u, v) = \frac{\sum_i |u_i - v_i|}{\sum_i |u_i + v_i|}$
2. Canberra: $dist(u, v) = \sum_i \frac{|u_i - v_i|}{|u_i| + |v_i|}$
3. Chebyshev: $dist(u, v) = \max |u_i - v_i|$
4. Euclidean: $dist(u, v) = \sqrt{\sum_i (u_i - v_i)^2}$
5. Manhattan: $dist(u, v) = \sum_i |u_i - v_i|$

4 Dataset

The data we are using is of US politician's tweets, up until May of 2017 [3]. Due to twitter API restrictions, for frequent tweeters we only have their last 3200 tweets (until May, 2017) while for others we have all their tweets. The dataset comprises over 1.2 million tweets from 545 US politicians (Presidents, Congress and Governors). From this data, we drop any author who has less than 500 tweets so that each author has a minimum number of tweets to train with. This pruning of infrequent authors left us with 500 authors remaining in our dataset.

[3]The choice of hashing down the Bag of Words vector to 300 was based on the dimensionality of the GloVe embeddings.

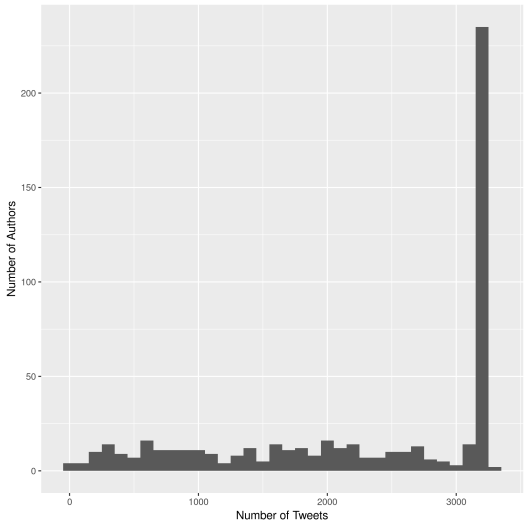

Figure 1: Histogram showing the skewed distribution of tweets per author.

4.1 Exploratory analysis and dropping fields

Starting with the tweet data, we looked at the various fields that are available in the data and removed those that are not useful for the author attribution task. Out of the 24 available fields, we initially retained only 7 that could have been potentially useful for classification including number of retweets, number of favorites, timestamp, etc. However, after some initial experiments the results indicated that the performance of the models wasn't affected regardless of whether these extra fields were given as input or not along with the tweet text. In our final experiments, we drop all fields other than the text of the tweet. This also allows us to compare the features extracted from the text in a pure form without the other fields helping/interfering with the learning models' performance.

4.2 Splitting into different regimes

Our initial experiments were run with the whole dataset after pruning infrequent authors and the classifiers had to distinguish 500 classes. However, the models took too long to train (over a day) and they performed very poorly (accuracy $\approx 3\%$). This forced us to reduce the number of classes significantly.

Figure 1 shows the histogram of tweets per author which has a clearly skewed distribution. Due to this, we pick authors/classes from two different regimes: infrequent tweeters, who have tweeted less than 1000 times, and frequent tweeters, who have tweeted at least 3000 times. Out of these two regimes, we pick the 10 or 20 most frequent tweeters (top 10 and top 20 tweeters). Table 5 in the Appendix contains statistics about these dataset splits.

4.3 Experiments

The final experiments (in each regime) were all run on a random 5-fold cross validation split of the data, and all reported results are the average classification accuracy across all folds. There are 5 different features that we use:

1. Explicit Linguistic Features (Ling-Feat)
2. Bag of Word bigrams (BoW-2Hash)
3. Bag of Word trigrams (BoW-3Hash)
4. SIF embeddings (SIF)
5. SIF embeddings after removing the principle component (SIF-PC)

Each of these features were given as input to the 13 models described in Section 3.4. This gives us 65 different feature-model combinations.

The training was split across two machines with the following specs:

- Smaller machine: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 12 cores, L1 cache: 32K, L2 cache: 256K, L3 cache: 12288K, RAM: 32GB
- Bigger machine: Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz, 72 cores, L1 cache: 32K, L2 cache: 1024K, L3 cache: 25344K, RAM: 256GB

The Neural Network models often needed the bigger machine to train quickly while many of the other models could often be trained in parallel on the smaller machines. Reducing the number of classes (and hence the size of the data) significantly affected the scalability of our learning models. These models were prohibitively slow with 500 classes and performed poorly. Even on the smaller datasets, due to 65 different training configurations and 5-fold cross validation, the complete training process took more than a day to finish.

4.3.1 Reproducibility

A key detail through all our implementation is the focus on reproducibility of the results. We use Makefiles with commands defined to download, split, process the data and also to train and evaluate the models. Variables that affect the final accuracy of the models (particularly the random state used) are exposed through the Makefile and are set by default to our settings to get exactly matching results.

5 Quantitative Results and Qualitative Analysis

Table 1: Classification Accuracy for top 10 authors ($\leq$ 1000 tweets)

Models/Features	Ling-Feat	BoW3-Hash	BoW2-Hash	SIF	SIF-PC
RandomForest	22.06	26.02	33.76	31.68	32.11
SVM	12.42	28.08	36.98	62.29	62.13
GradientBoosting	26.99	32.96	41.54	53.10	52.75
NN	17.15	25.36	33.83	62.00	62.40
KNN-Canberra	21.75	13.60	14.58	49.06	50.43

Table 2: Classification Accuracy for top 20 authors ($\leq$ 1000 tweets)

Models/Features	Ling-Feat	BOW3-Hash	BOW2-Hash	SIF	SIF-PC
RandomForest	12.39	19.54	26.86	20.10	19.82
SVM	5.33	21.73	29.14	53.00	52.86
GradientBoosting	17.15	25.27	34.20	41.05	40.78
NN	11.75	15.44	23.12	49.32	49.55
KNN-Canberra	12.37	9.15	10.35	37.31	38.95

Table 3: Classification Accuracy for top 10 authors ($\geq$ 3000 tweets)

Models/Features	Ling-Feat	BOW3-Hash	BOW2-Hash	SIF	SIF-PC
RandomForest	25.24	29.23	35.96	35.98	35.91
SVM	11.47	29.83	37.32	65.01	64.88
GradientBoosting	29.83	33.65	42.40	56.54	56.28
NN	22.51	26.31	33.91	67.90	68.40
KNN-Canberra	25.35	18.27	20.17	54.78	56.36

An overview of the results, using the most interesting learning models, for the four regimes are shown in Tables 1, 2, 3, 4. Detailed results for all learning models can be found in the Appendix (Tables 6,

Table 4: Classification Accuracy for top 20 authors ($\geq$ 3000 tweets)

Models/Features	Ling-Feat	BOW3-Hash	BOW2-Hash	SIF	SIF-PC
RandomForest	12.57	16.69	22.17	21.41	21.21
SVM	5.80	17.77	24.89	49.75	49.58
GradientBoosting	16.92	20.92	29.32	40.66	40.77
NN	15.25	16.00	22.90	51.29	50.87
KNN-Canberra	12.72	9.80	11.50	39.59	41.04

7, 8, 9). All the values listed are average of percentage classification accuracy across a 5-fold cross validation that the models achieve. In all the tables as well as the subsequent analysis, Ling-feats is Linguistic features, BoW2-Hash is Bag of Word bigrams, BoW3-Hash is Bag of Word trigrams, SIF is SIF embeddings and SIF-PC is SIF embeddings after removing the principle component. Further, sometimes BoW is used to mean both bigrams and trigrams.

5.1 Qualitative analysis of the results

In all four regimes, Ling-Feats perform the worst. This provides empirical evidence for the hypothesis from Section 3.1. A tweet has too little text for the explicit linguistic features to provide any statistically significant information that can be used by the learning models for classification. The only exception is the KNN classifier with Canberra distance metric in which classification accuracy using Linguistic Features outperforms BoW. However, compared to the SIF features, even this accuracy is poor. This is most likely an artifact of the way the Canberra distance metric works which allows it to infer more from the smaller dimensional linguistic features as compared to the 300 dimensional BoW features.

The accuracy using BoW is generally between those of Ling-Feats and SIF. However, in the case of Random Forest, we observe that BoW2-Hash performs better or comparable to SIF. A possible reason for this is that BoW2-Hash has discrete entries in each of its dimensions which the random forest classifier can handle better than the real values of SIF.

Another trend in the BoW results is the consistent inferior performance of the BoW3-Hash compared to BoW2-Hash. This happens because BoW3-Hash are much more powerful in terms of their expressiveness which causes the learning algorithms to overfit on the training dataset resulting in the poor accuracy in the test dataset. The hypothesis of overfitting due to expressiveness is further supported by the fact that all learning models give an inferior performance with BoW3-Hash.

The best overall performance is generally achieved with latent space SIF embeddings. Except for a few minor exceptions (Random Forests giving better performance with BoW2-Hash), we observe that SIF and SIF-PC embeddings provide highest accuracy across all models in all four datasets. Comparing SIF and SIF-PC also leads to some interesting observations. SIF-PC is obtained by removing the principal component from the SIF embedding which should remove information about grammatical structure and stop words resulting in more meaningful embeddings and hence in better performance. However the theoretically expected behaviour doesn't always manifest in emipirical results. With some learning models SIF performance slightly better while with certain other models SIF-PC performs better. And these differences are too small to be of any practical significance.

In terms of the learning models, we note that either the SVM or the NN perform best with either SIF or SIF-PC, with slight variations across different datasets. SVMs also have another interesting aspect wherein they sometimes get the best performance overall with SIF/SIF-PC but they also consistently perform the worst when given the Ling-Feats as input.

5.2 Changing number of classes

We now look at the effect of increasing the number of authors with similar number of tweets per author. Specifically, we compare the classification accuracy of configurations in 3 with 4 and those of 1 with 2. In both of these cases, the number of authors are doubled, thus doubling the number of

classes for the model to choose from. As expected, we observe a uniform decrease in the classification accuracy of all models for all features without any exceptions.

5.3 Changing number of data points per class

We now analyze the effect of increasing the number of tweets per author. Specifically, we compare the classification accuracy of configurations in 1 with 3 and those of 2 with 4. In both of these cases, the average number of tweets per author is increased by more than 3 times. Intuitively, all configurations are expected to perform better since there is more training data to develop a more precise model. However this is empirically seen only in the case of datasets with 10 authors. All configurations perform better with more tweets per author with the only exception being SVM with Linguistic Features which performs marginally better with fewer tweets. In the case of 20 authors (Tables 2 and 4) we observe that the performance of most models trained with BoW tends to worsen with more tweets per author. This can be attributed to there being more collisions while hashing.

6 Conclusions

We analysed the performance of various feature extraction/engineering methods for the task of author attribution with very short texts as input. Our results show that the smart and simple latent space embedding techniques outperform the combinatorial brute force like Bag of Words representations which also need clever engineering tricks to become scalable. Further, we showed that classically used explicit linguistic features fail to perform without a large corpus of text per author. Our empirical results also show that this task is inherently difficult when there are more authors/classes.

Our experimental setup also showed that it is important to have an easily reproducible and repeatable pipeline to starting from processing the data to finally training and evaluating the models. This allowed us to train 13 models on 5 feature sets with 4 datasets for a total of 260 configurations. A robust pipeline allowed using multiple machines and iterating through failed attempts to quickly incorporate feedback to improve our models and results.

6.1 Future Work

Our results open up multiple interesting questions. The first and foremost is the question of scaling with number of authors/classes. One way to address this might be to train one model per class and output the final prediction based on the confidence of individual models. Another approach would be to add more features which might help give the models more discriminative power to tackle more classes. Another direction is to try more complex version of existing models, for example more complex sentence embedding methods, utilizing deeper neural networks, or increasing the dimension of the Bag of Word ngrams.

References

[1] Sanjeev Arora, Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. A compressed sensing view of unsupervised text embeddings, bag-of-n-grams, and lstms. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[2] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. *International Conference on Learning Representations*, 2017.

[3] Stuck_In_the_Matrix (Jason Baumgartner). *Over one million tweets collected from US Politicians (President, Congress and Governors)*, 2017 (accessed October 5, 2018). URL: `https://www.reddit.com/r/datasets/comments/6fniik/over_one_million_tweets_collected_from_us/`.

[4] Esteban Castillo, Darnes Vilariño, Ofelia Cervantes, and David Pinto. Author attribution using a graph based representation. In *Electronics, Communications and Computers (CONIELECOMP), 2015 International Conference on*, pages 135–142. IEEE, 2015.

[5] Carole E Chaski. Who's at the keyboard? authorship attribution in digital evidence investigations. *International journal of digital evidence*, 4(1):1–13, 2005.

[6] Luke Chen Coline Nantermoz, Eric Gonzalez. *Authorship Attribution with Limited Text*, 2017 (accessed October 5, 2018). URL: `http://cs229.stanford.edu/proj2017/final-reports/5241953.pdf`.

[7] Olivier De Vel, Alison Anderson, Malcolm Corney, and George Mohay. Mining e-mail content for author identification forensics. *ACM Sigmod Record*, 30(4):55–64, 2001.

[8] Tom Kenter, Alexey Borisov, and Maarten de Rijke. Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv preprint arXiv:1606.04640*, 2016.

[9] Vlado Kešelj, Fuchun Peng, Nick Cercone, and Calvin Thomas. N-gram-based author profiles for authorship attribution. In *Proceedings of the conference pacific association for computational linguistics, PACLING*, volume 3, pages 255–264, 2003.

[10] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[11] Kim Luyckx and Walter Daelemans. Authorship attribution and verification with many authors and limited data. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 513–520. Association for Computational Linguistics, 2008.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[14] F. Mosteller and D. L. Wallace. Inference and disputed authorship: The federalist. *Review of the International Statistical Institute*, 34(2):277–279, 1964.

[15] S Ouamour and Halim Sayoud. Authorship attribution of short historical arabic texts based on lexical features. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on*, pages 144–147. IEEE, 2013.

[16] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*, 2017.

[17] 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. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

[19] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[20] Reza Ramezani, Navid Sheydaei, and Mohsen Kahani. Evaluating the effects of textual features on authorship attribution accuracy. In *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*, pages 108–113. IEEE, 2013.

[21] Conrad Sanderson and Simon Guenter. Short text authorship attribution via sequence kernels, markov chains and author unmasking: An investigation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 482–491. Association for Computational Linguistics, 2006.

[22] Wikipedia contributors. The cuckoo's calling — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=The_Cuckoo%27s_Calling&oldid=862162319`, 2018. [Online; accessed 5-October-2018].

A Appendix

Table 5: Description of the Dataset

Dataset/Description	Number of Authors	Number of Tweets	Average number of Tweets
Whole Dataset	545	1242871	2280.5
After pruning infrequent authors	500	1230492	2461.0
Top 10 with ≥ 3000 tweets	10	32495	3249.5
Top 20 with ≥ 3000 tweets	20	64946	3247.3
Top 10 with ≤ 1000 tweets	10	9522	952.2
Top 20 with ≤ 1000 tweets	20	18072	903.6

Table 6: Classification Accuracy for top 10 authors (≤ 1000 tweets)

Models/Features	Ling-Feat	BoW3-Hash	BoW2-Hash	SIF	SIF-PC
RandomForest	22.06	26.02	33.76	31.68	32.11
Gaussian-NaiveBayes	18.44	24.86	31.25	35.85	35.39
SVM	12.42	28.08	36.98	62.29	62.13
GradientBoosting	26.99	32.96	41.54	53.10	52.75
AdaBoost	23.61	23.44	30.58	34.30	34.03
Bagging	21.68	27.98	36.70	35.68	34.96
ExtraTrees	21.53	25.55	33.40	31.42	30.73
NN	17.15	25.36	33.83	62.00	62.40
KNN-Euclidean	22.31	14.05	16.39	45.68	47.60
KNN-Manhattan	22.24	13.26	14.46	47.94	48.40
KNN-Braycurtis	22.21	28.46	37.26	50.78	51.89
KNN-Chebyshev	22.11	11.80	14.51	30.21	33.43
KNN-Canberra	21.75	13.60	14.58	49.06	50.43

Table 7: Classification Accuracy for top 20 authors ($\leq$ 1000 tweets)

Models/Features	Ling-Feat	BOW3-Hash	BOW2-Hash	SIF	SIF-PC
RandomForest	12.39	19.54	26.86	20.10	19.82
Gaussian-NaiveBayes	11.79	18.31	24.25	26.71	26.56
SVM	5.33	21.73	29.14	53.00	52.86
GradientBoosting	17.15	25.27	34.20	41.05	40.78
AdaBoost	13.56	16.65	20.62	21.51	21.34
Bagging	12.43	21.77	29.04	23.13	22.40
ExtraTrees	12.22	19.13	25.67	18.29	18.64
NN	11.75	15.44	23.12	49.32	49.55
KNN-Euclidean	12.33	9.87	12.02	34.61	35.54
KNN-Manhattan	12.20	9.22	10.12	36.03	36.38
KNN-Braycurtis	12.25	20.96	29.57	38.84	40.47
KNN-Chebyshev	12.47	6.48	7.44	19.26	22.37
KNN-Canberra	12.37	9.15	10.35	37.31	38.95

Table 8: Classification Accuracy for top 10 authors ($\geq$ 3000 tweets)

Models/Features	Ling-Feat	BOW3-Hash	BOW2-Hash	SIF	SIF-PC
RandomForest	25.24	29.23	35.96	35.98	35.91
Gaussian-NaiveBayes	22.70	27.10	32.47	35.01	35.34
SVM	11.47	29.83	37.32	65.01	64.88
GradientBoosting	29.83	33.65	42.40	56.54	56.28
AdaBoost	26.81	24.75	29.52	34.63	34.41
Bagging	25.23	30.90	39.21	39.36	39.27
ExtraTrees	24.92	28.94	35.11	34.30	34.37
NN	22.51	26.31	33.91	67.90	68.40
KNN-Euclidean	25.38	19.35	22.40	53.21	55.52
KNN-Manhattan	25.23	18.62	20.57	55.14	56.09
KNN-Braycurtis	25.23	31.62	39.85	57.01	58.28
KNN-Chebyshev	25.30	13.21	13.70	35.96	40.43
KNN-Canberra	25.35	18.27	20.17	54.78	56.36

Table 9: Classification Accuracy for top 20 authors ($\geq$ 3000 tweets)

Models/Features	Ling-Feat	BOW3-Hash	BOW2-Hash	SIF	SIF-PC
RandomForest	12.57	16.69	22.17	21.41	21.21
Gaussian-NaiveBayes	11.98	15.98	21.55	22.93	23.23
SVM	5.80	17.77	24.89	49.75	49.58
GradientBoosting	16.92	20.92	29.32	40.66	40.77
AdaBoost	14.48	13.93	17.94	21.43	21.56
Bagging	12.60	17.65	24.73	23.40	23.44
ExtraTrees	12.66	16.22	21.21	20.10	20.34
NN	15.25	16.00	22.90	51.29	50.87
KNN-Euclidean	12.52	10.57	12.81	37.68	39.49
KNN-Manhattan	12.43	9.97	11.57	39.07	39.90
KNN-Braycurtis	12.45	19.23	26.51	41.17	43.01
KNN-Chebyshev	12.34	6.87	7.66	22.95	26.44
KNN-Canberra	12.72	9.80	11.50	39.59	41.04