Whose Book is it Anyway?
Using Machine Learning to Identify the Author of Unknown Texts

Sean Stanko, Devin Lu, Irving Hsu

Computer Science Department

Stanford University, Stanford, CA 94305

{ststanko, devinlu, irvhsu}@stanford.edu

Abstract

In this paper, we present an implementation of a new method for text classification that achieves significantly faster results than most existing classifiers. We extract features that are computationally efficient and forgo more computationally expensive ones used by many other text classifiers, most notably *n*-grams (contiguous sequences of *n* words). We then analyze the feature vectors through a hybrid SVM technique that sequentially classifies them with a one-versus-all classifier and a sequence of binary classifiers. Our overall method finishes execution within seconds on large novels, with accuracy comparable to that of standard classifiers but with a significantly shorter runtime.

1. Introduction

Identifying the author of a text is a real, and recurring, problem that reappears again and again across many different fields. Archaeologists and historians regularly recover and attempt to identify unattributed texts. Academics are constantly on the lookout for plagiarism. Author attribution even appears in popular culture, as with *The Cuckoo's Calling* incident this summer, when the author "Robert Galbraith" was revealed to be a pseudonym for J.K. Rowling.

Naturally, this is a very active, and lucrative, field of machine learning. Part of the reason for its activity is that it is, unsurprisingly, both a difficult and somewhat open-ended task, with numerous approaches and success rates.

1.1 Approach

In relation to this field of study, we will create a machine learning system that will attribute a provided text to one of a series of known authors; more specifically, we will be attempting to do so with novels. Based on existing literature and own personal experience in the field, text analysis often has less-than-stellar runtimes [1]; to this effect, we also plan to design our classifier to complete with a reasonably fast runtime.

To go about this, we plan to use a combination of shallow text analysis and supervised machine learning. Our algorithm will be a two-step multiclass SVM, which performs a series of one-versus-all SVMs on an input and then a second "runoff" series of binary classifiers on all possible positive labels from the one-versus-all. With our inputs potentially having large feature vectors (specifically, 109 features each), and the need to run several instances of binary classification, the SVM's ability to efficiently perform classifications via the kernel trick is ideal for this scenario.

We will be parsing our input texts as .txt files, courtesy of Project Gutenberg (www.gutenberg.org). The text analysis, as well as the construction of feature vectors, will be done in Python and then saved to a second .txt file. Finally, training and classification using these features will be performed in MATLAB.

2. Text Analysis/Feature Extraction

2.1 Overview

One of our first, and key, observations we made about our training set of novels is that they are generally long, and likewise have rich vocabularies and styles. From this, we then made our key assumption for our training algorithm:

Each author has a "true," statistically-representable style that their works are slight variations on.

With these observations and this assumption in mind, we researched existing literature for leads on potential methodologies. Almost immediately we came across, and chose to pursue the statistically-based shallow text analysis approach (as opposed to the more semantically based deep text analysis) [2], as it is well-aligned with our assumption of a statistically based author style and, more importantly, is significantly less expensive to compute. Further research into shallow text analysis suggested three main approaches:

Token Based: Word-level features such as word length and *n*-grams.
Vocabulary Based: Vocabulary richness indicators, such as the number of unique words.
Syntactic Features: Sentence-based features, such as parts of speech and punctuation.

Each of these feature approaches have their benefits and drawbacks; token level features are easy but, especially in the case of *n*-grams (contiguous sequences of *n* words), slow to compute. Vocabulary features are both fast and easy to compute, but can have a high variance and are thus inaccurate for small texts (approx. 1000 words or less). Syntactic features share many of their qualities with vocabulary features, but can be quite complex to compute [2].

2.2 Language Model

With our goal of achieving a fast runtime, we decided to take an original approach to our language model. Given the long nature of our sample texts (most books run at least 100,000 words in length), we opted to forgo computationally expensive features (such as *n*-grams) altogether and instead focused on easily computable features, particularly those pertaining to vocabulary and syntax. Just as the long lengths of our texts make the expensive features unappealing, they also make the cheaper ones powerful; as mentioned, vocabulary and syntactic features scale in accuracy and precision with text size.

The Legomena and Richness rates are standard vocabulary indicators, and word and sentence lengths are both standard, easily computed token and syntactic features [2]. We decided to look at nominative pronouns and conjunctions as they both have small word pools, but are still powerful in determining sentence structure, representing subject and clause shifts respectively.

Our final feature vector design treats each book as an individual training example and consists of 109 features split across the seven categories listed in Table 1. Note that, aside from a check against a small pool of conjunctions and pronouns, we do not consider the semantic meaning of any individual word; we are only concerned with a word's length and uniqueness. This enables us to achieve a very low runtime in our feature extractor, of approximately 11 seconds for 37 books (~4 million total words).

Table 1. Summary of features used in the language model. All distributions are represented as counters for values 0-25 and an additional value for 26+.

Feature	Description
Hapax Legomena[1]	Number of words that occur exactly once.
Dis Legomena[1]	Number of words that occur exactly twice.
Vocabulary Richness[2]	Total number of unique words.
Sentence Length Distribution[3]	Lengths of sentences in the text.
Word Length Distribution[2]	Lengths of the words in the text.
Pronoun Distribution[3]	Occurrences of nominative pronouns per sentence.
Conjunction Distribution[3]	Occurrences of conjunctions per sentence.

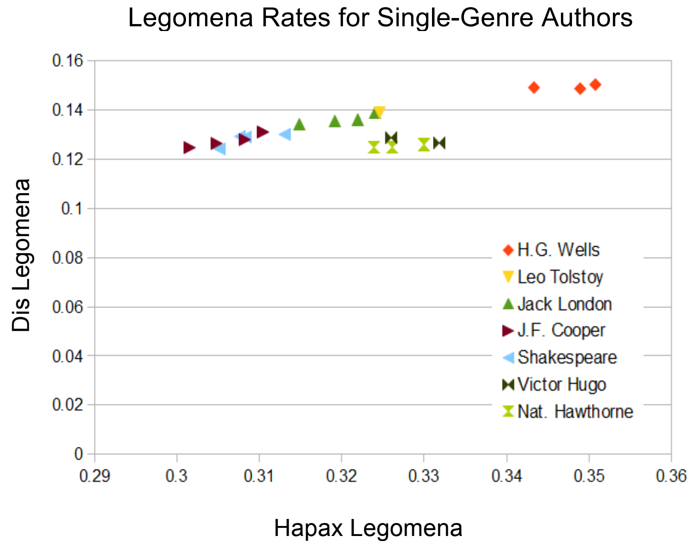


Figure 1. Legomena rates for a subset of our authors. Note that, as in our governing assumptions, the values for various works appear to be tightly clustered around an average for that author.

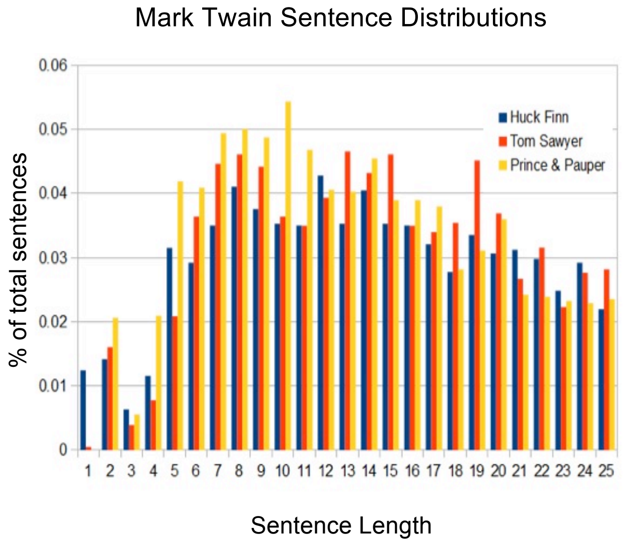


Figure 2. Sentence distributions for select works by Mark Twain.

[1] Normalized by number of unique words.
[2] Normalized by number of words.
[3] Normalized by number of sentences.

3. Classification Methodology

Once the features were extracted, we executed a hybrid multi-classification strategy to determine the author of each test sample. The test classification proceeded in two steps. First, for each author *k* a one-versus-all classifier that could test whether a given example was written by that author was created. Each classifier was created by training an SVM on the entire training set with training examples given a positive label if they were written by author *k* and a negative label otherwise. A test sample was run through each of these classifiers, and if exactly one classifier returned a positive result the test sample was assigned the corresponding author.

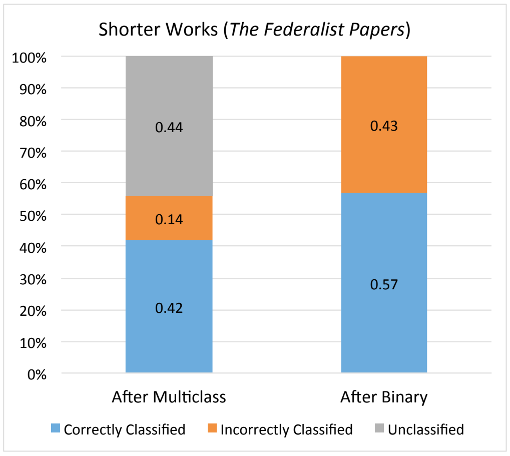
If the one-versus-all classification resulted in zero or more than one positive results, we moved on to a series of binary classifiers. A binary classifier for authors *n* and *m* was created by training an SVM on the training examples of authors *n* and *m* for all distinct pairs (*n*, *m*). The test sample was then run through each of these classifiers. If the test sample was attributed to a certain author *i* against all other authors $j \neq i$, then it was assigned to author *i*. Note that because of the symmetry of binary classification, this procedure can produce at most one assignment. If neither of these procedures resulted in an assignment for the test sample, then it was left unattributed.

4. Results and Analysis

We tested our method on two training/test sets: a set of longer works (full-length novels), which included 27 training examples and 10 test examples from 8 separate authors, and a set of shorter works (the Federalist Papers), which included 38 training examples and 7 test examples from 3 separate authors.

Our method resulted in approximately 70% true positive classification for novels and approximately 57% true positive classification for the Federalist Papers. In general, the multi-class step was less likely to successfully attribute a test sample, but was also less likely to incorrectly attribute a test sample.

Our method saw a considerable performance loss with shorter works, as it achieved only 57% correct classification out of a universe of three authors. However, with longer works, our method performed quite well, achieving a 70% correct classification with a universe of eight authors. This is slightly worse than typical classifiers that include *n*-gram features, which typically have a test accuracy of approximately 85% [1]. However, our method demonstrated a substantial advantage in runtime: we trained and tested all of our novels in approximately 15 seconds, and the training and testing for the Federalist Papers was too fast to meaningfully measure. This contrasts with runtimes that are on orders of minutes or hours for typical *n*-gram classifiers used on novels.



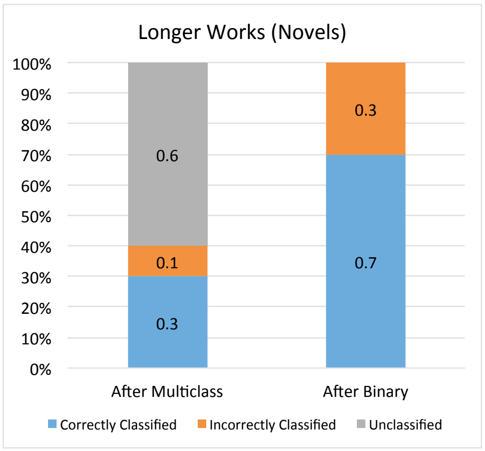


Figure 3. Hybrid multi-classification results for both short works (the Federalist Papers) and long works (novels).

In general, adding training examples quickly reduced the number of unclassified test examples to zero. Furthermore, adding more training examples tended to decrease our testing error. This suggests that we are not overfitting our model – in fact, further performance improvements may be attained by including more training examples.

5. Conclusion

Overall, we are satisfied with the results of our classifier. While our accuracy is slightly below that of other attribution systems, it is still well within an acceptable range and more than makes up for any shortcomings with its phenomenal runtime. Furthermore, we suspect that, given a few slight improvements, we can increase our accuracy without changing the speed or computationally efficient nature of our algorithm.

The most obvious step for further work is to find additional training examples. We were constrained by the works that could be found on Project Gutenberg, and as such some authors, such as Tolstoy, had an insufficient number of training texts. Our learning curve suggests that acquiring additional training data will produce positive improvement.

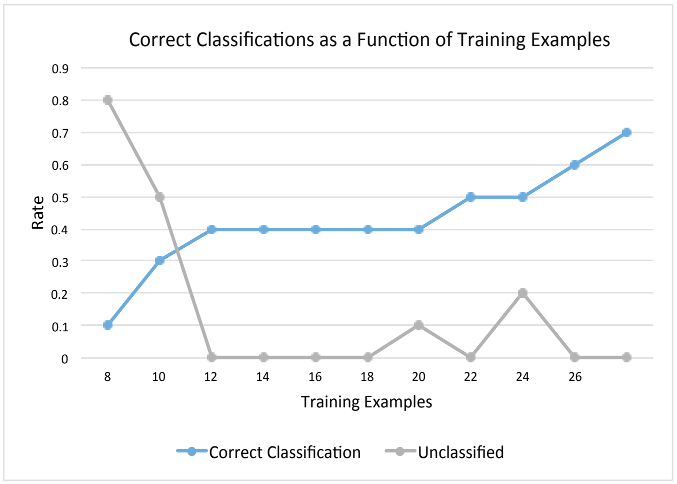


Figure 4. Learning curve corresponding to the percentage of correct classifications as a function of the number of training examples.

Additionally, since there is little evidence of overfitting and our runtimes are very low, expanding the feature vector could produce gains. Our feature extractor is a linear parser (it linearly scans each text for the given features), so it should be possible to introduce more easily-computable features with minimal risk of introducing overfitting or significantly hindering our runtime.

6. Acknowledgements

The authors would like to thank Professor Andrew Ng and the CS229 course staff for their teaching and support in this class and on our project. We would also like to thank the developers of the Python regex library and the LIBLINEAR SVM library for greatly simplifying the process of implementing a sentence parser and SVM, respectively.

7. References

[1] Sagae, K. and Lavie, A. A Classifier-Based Parser with Linear Run-Time Complexity. Language Technologies Institute, Carnegie Mellon Unviersity. 2005. http://www.cs.cmu.edu/~alavie/papers/IWPT05-sagae.pdf

[2] Luyckx, K. and Daelemans, W. (2005). Shallow Text Analysis and Machine Learning for Authorship Attribution. In Proceedings of the Fifteenth Meeting of Computational Linguistics in the Netherlands. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.5550&rep=rep1 &type=pdf

[3] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.R., and Lin, C.-J. LIBLINEAR: A Library for Large Linear Classification, Journal of Machine Learning Research 9(2008), 1871-1874. Software available at http://www.csie.ntu.edu.tw/~cjlin/liblinear