Comparing Deep Learning and Conventional Machine Learning for Authorship Attribution and Text Generation

Gregory Luppescu
Department of Electrical Engineering
Stanford University
gluppes@stanford.edu

Francisco Romero
Department of Electrical Engineering
Stanford University
faromero@stanford.edu

Abstract

The classic problem of authorship attribution has been thoroughly explored with conventional machine learning models, but has only recently been studied using state-of-the-art neural networks. In this paper, we investigated two tasks: generating text given a particular author's style of writing, and classifying a document by its respective author. We used a Long Short Term Memory neural network for both tasks, and we implemented conventional machine learning techniques as baselines for the author classification task. We also trained our models using a window of words as opposed to an entire document. Our generative model produced a wide range of results across all authors, only limited by the dataset size. For the classification models, conventional machine learning methods performed reasonably, but were outperformed by the deep learning model.

1 Introduction

Authorship attribution is a well-studied task that seeks to answer the question: *which select author wrote a given document, novel, or text?* As described in [1], the motivation behind authorship attribution is to use linguistic patterns and markers to identify authors by their work. For humans, this can quickly lead to a daunting task as the number of candidate authors grows and the textual data becomes longer. This makes the task suitable for machine learning and, more recently, neural network based models. Authorship attribution also has real world applications for automatically labeling text and finding similarities between authors' writings, the latter of which, for example, can be used for generating personalized customer recommendations and suggestions.

In this project, we developed and assessed two different models that center around the authorship attribution task. First, we created a classification model that attempted to tie a corpus of text to a particular author. For this model, we compared Multinomial Naïve Bayes (NB) and Gaussian Discriminant Analysis (GDA) to a Long Short Term Memory (LSTM) neural network. Second, we used an LSTM neural network to create a generative model that produces words given the learned style of an author. This generative model could serve as the basis for a future study in language style transfer.

2 Background and Related Work

The first seminal authorship attribution study was done by Mosteller and Wallace with the Federalist Papers [2]. The task has also been widely studied using conventional machine learning techniques, such as NB. Sebastiani was one of the first to study this task, and used NB and Support Vector Machines on Reuters to achieve 81.5% and 92% F_1-scores, respectively, across 10 categories [3].

Peng *et al.* combined NB with statistical language models to achieve an accuracy of 96% on a dataset of 10 Greek authors [4]. Recently, Ge *et al.* demonstrated that neural network language models can also achieve high accuracy (80%) when applied to authorship attribution tasks [5]. Finally, generating text using neural networks has been explored by Sutskever *et al.*, in which they used a variant of the recurrent neural network (RNN) model called the multiplicative RNN (MRNN) [6]. Sutskever *et al.* performed a quantitative and qualitative analysis of their MRNN's ability to generate text in various different contexts (e.g. from Wikipedia data).

This project differs from the aforementioned studies in that we used fiction novels, poems, and scientific works for our experiments. We also used a more modern, state-of-the-art network that features LSTM cells. Finally, for the classification models, we predicted across 10 authors, and we used a window of words as opposed to an entire document.

3 Approach

3.1 Dataset and Data Pre-processing

To perform our experiments, we used a subset of the Project Gutenberg Dataset, which contains 3,036 books written by 142 authors [7]. We selected 10 authors and an equal number of lines per author (12,000) across all of their available works. To select the authors, we considered their writing era and their text genre. The 10 authors included Charles Darwin, Edgar Allan Poe, Edward Stratemeyer, Jacob Abbott, Lewis Carroll, Mark Twain, Michael Faraday, Ralph Waldo Emerson, Rudyard Kipling, and Winston Churchill.

To prepare the data for use in our models, we developed a parser that performed the following tasks:

- Count all tokens (words and punctuation) in the dataset
- Convert all tokens that fall below a count threshold to the "unknown" (`<unk>`) token
- Replace names with the "name" (`<name>`) token
- Convert all words to lowercase
- Separate punctuation from words (e.g. apples, bananas, and oranges. $\rightarrow$ apples , bananas , and oranges .)
- Put all sentences onto unique lines

To filter names, we used a name database provided by [8]. Note that names such as *An*, *My*, and *Man* were excluded from being filtered, as they are usually not used as names.

3.2 Language Model and Word Embeddings

A language model is a probability distribution over a sequence of words. For authorship attribution and text generation, our language model is given as:

$$LM = \mathrm{P}(X|w_t, w_{t-1}, ..., w_1)$$

where X represents either an author (when used for author classification) or the next word in the sequence (when used for text generation), and w_t represents a word at position t. To learn these probability distributions, words must be represented by computable quantities. This can be achieved by representing the vocabulary in an embedding space with the hope that the vector representations can capture and disambiguate semantic properties of words. As a result, for most of our experiments, we utilized the GloVe word embeddings provided by [9]. We used GloVe word embeddings over word2vec because GloVe captures more information and better utilizes the statistics of the word vectors by using a global co-occurrence matrix. Contrarily, word2vec uses separate local context windows, which could lead to a less descriptive and less effective word vector representation.

3.3 Classification using Conventional Machine Learning

As a baseline for our deep learning authorship attribution task, we implemented classification models using NB and GDA. For all models, a single training example was represented by the pair (x, y),

where x is a sequence of words of length equal to a specified window length, and y is a label indicating the author who wrote the text from x. In addition, for both models, the words in x were each subsequently transformed to their appropriate GloVe embedding representation. We chose to combine the GloVe vectors in each training example with a Bag of Words approach by making each training example be the sum of its word vectors. For each of the aforementioned classification methods, the only hyperparameter modified was the window size per training example. We selected the window size that maximized the accuracy on the development set of the data.

3.4 Classification using Deep Learning

To classify the authors, we also used an LSTM-based network in which each training example is a window of words associated with a particular author label. The labels are represented as 10-dimensional one-hot vectors, where the non-zero entry corresponds to the target author.

The LSTM is described by the equations below:

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right), \quad f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right), \quad o_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

$$\tilde{c}_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right), \quad c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad h_t = o_t \odot \tanh(c_t)$$

where i_t is the input gate, f_t is the forget gate, o_t is the output gate, $\tilde{c}_t$ is the new memory cell, c_t is the final memory cell, h_t is the final hidden state, and $\sigma(\cdot)$ is the sigmoid function. The intuition behind the LSTM cell is that the input gate determines how much the current cell matters, the forget gate determines how much the past matters, and the output gate determines how much the cell is exposed. LSTM networks help with the vanishing gradient problem because they are better at "remembering" information from the past. Since we are using a window of words, past information is crucial to the performance of our classification and generation tasks, hence the reason why we selected an LSTM-based model. A corresponding illustration of an LSTM cell can be seen in Figure 1.

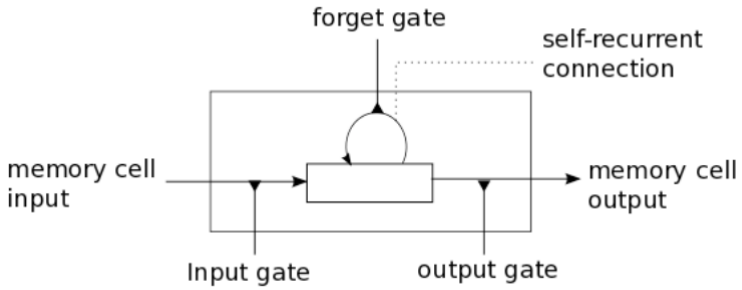

Figure 1: Example LSTM cell

Our model also utilizes Dropout to prevent overfitting. Dropout is a form of regularization that works by randomly setting units in the hidden layers to zero with some probability. This technique forces the network to learn multiple independent representations of the same data, which has been shown to help the model generalize more effectively. Finally, our model incorporates gradient clipping, which helps to mitigate the effects of "exploding" gradients.

3.5 Text Generation using Deep Learning

The text generation model also utilizes an LSTM-based network (described in section 3.4) to predict the next word in the sequence for a specific author. A single training example is represented by the pair (x, y), where x is a sequence of words of length equal to a specified window length, and y is the next word in the sequence following x. For this project, we tried two different text generation tasks: a per-author text generator in which the sequences always come from a single author, and a mix-of-authors text generator in which each training example can come from a different author.

To minimize the text generation model, we used perplexity, which is given as:

$$\mathrm{PP}^{(t)}\left(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}\right) = \frac{1}{\bar{P}\left(\boldsymbol{x}_{\text{pred}}^{(t+1)} = \boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, ..., \boldsymbol{x}^{(1)}\right)} = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}}$$

The perplexity is the inverse probability of the correct word, according to the model distribution $\bar{P}$, where $\boldsymbol{y}^{(t)} = \boldsymbol{x}^{(t+1)}$ is the one-hot vector for the word at $t+1$, $\boldsymbol{x}^{(t)}, ..., \boldsymbol{x}^{(1)}$ is the "history" of words leading up to $\boldsymbol{x}^{(t+1)}$, $\hat{\boldsymbol{y}}^{(t)}$ is the probability distribution over the vocabulary, and $|V|$ is the size of the vocabulary. The magnitude of the sum in the denominator of the right-most equation is proportional to the "correctness" of the generated text. Thus, the lower the perplexity, the better the language model.

4 Experiments and Results

We selected 12,000 sentences from each author to be a part of the dataset. For all experiments, we used an 80-10-10 split of the dataset for the training, development, and testing sets, respectively. Note that we equally partition each author's sentences into the training, development, and testing sets to ensure there is no bias, which is especially important for the training set. Except for the LSTM-based classifier, we used the GloVe vectors found in `glove.42B.300d.txt` [9], which contains the GloVe vectors for 42 billion unique tokens in an embedding space of dimension 300. The LSTM-based classifier performed best by learning the embeddings.

4.1 Classification Baseline

For each of the conventional machine learning baseline methods, we created models using window sizes ranging from 10 words to 1000 words, and chose the window size that maximized the development accuracy. Both models were then trained using the determined optimal window sizes, and were subsequently used to evaluate the test set. Figures 2 and 3 show the training accuracy and development accuracy as functions of window size for each classification method, as well as the resultant confusion matrices when the models were evaluated on the test set. Table 1 shows accuracy results for each classifier using their optimal window sizes.

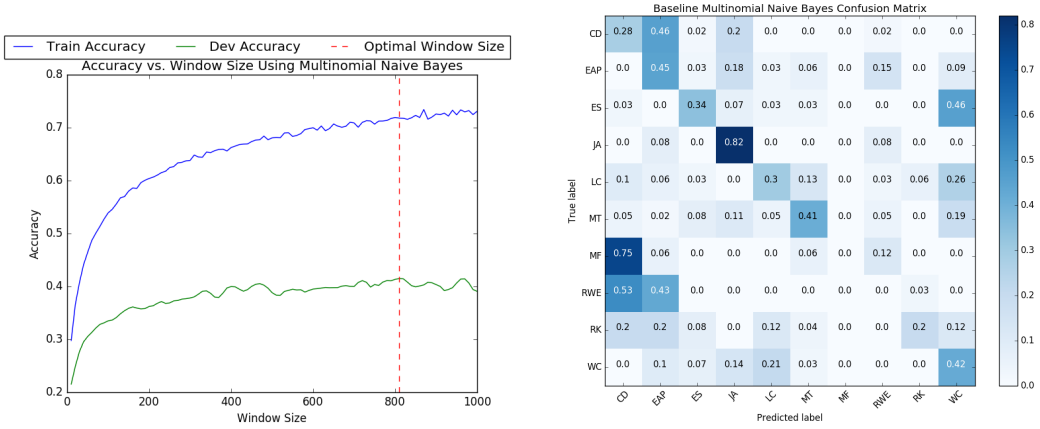

Figure 2: Training and development accuracies versus window size for NB classification (left) and the corresponding confusion matrix (right)

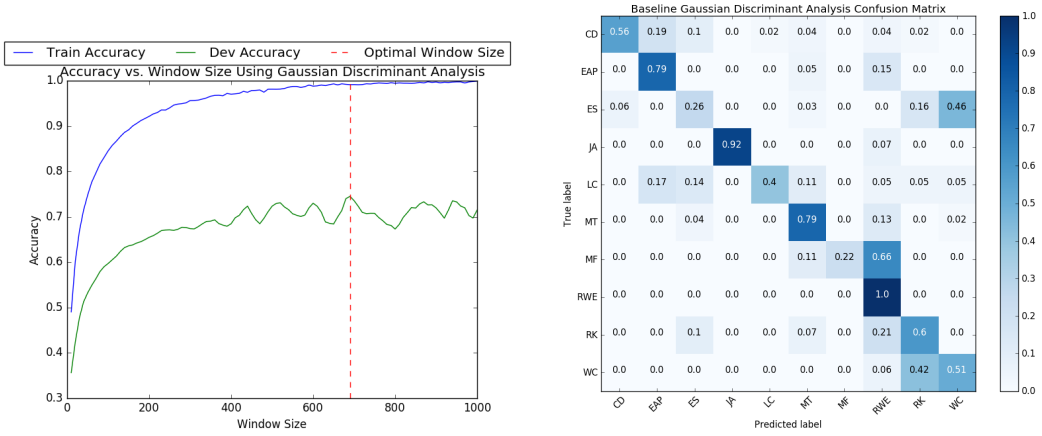

Figure 3: Training and development accuracies versus window size for GDA classification (left) and the corresponding confusion matrix (right)

Model	Window Size	Train Accuracy	Dev Accuracy	Test Accuracy
NB	810	0.721	0.415	0.368
GDA	690	0.990	0.745	0.656

Table 1: Accuracy results for the conventional machine learning classifiers

4.2 Classification using Deep Learning

The LSTM classifier used the hyperparameters listed in Table 2. The word window size of 200 was selected from a range of tested windows that ranged from 100 to 300. We used the Adam optimizer to minimize our loss. The training and development accuracies as functions of epoch, as well as the confusion matrix for the LSTM classifier are presented in Figure 5. The final training, development, and test accuracies were:

Training: 1.0 Development: 0.930 Test: 0.888

For reference, we created a simple baseline classifier using a linear combination of the input that is fed into a softmax unit. The baseline used an input window size of 400 words, as well as the same dataset sizes described in section 4. Despite being simple, the softmax classifier achieved a test accuracy of 77.8%. Thus, not only did the LSTM-based model surpass this classifier, it did so with a smaller window size. The confusion matrix in Figure 4 shows the results for the softmax classifier.

4.3 Text Generation

As described in section 3.5, we tried two different text generation tasks. First, we performed per-author text generation in which 10 separate models (one per author) were trained. Second, we

Parameter	Value
Learning Rate	0.001
Window Size	200
Number of Hidden Units	10
Number of Epochs	300
Keep Probability	0.5

Table 2: Hyperparameters used in Author Classification

Parameter	Value
Initial Learning Rate	1.0
Learning Rate Decay	0.8
Unrolled Steps of LSTM	35
Number of Hidden Units	650
Number of Epochs	20
Keep Probability	0.5
Vocabulary Size	50,000

Table 3: Hyperparameters used in Text Generation

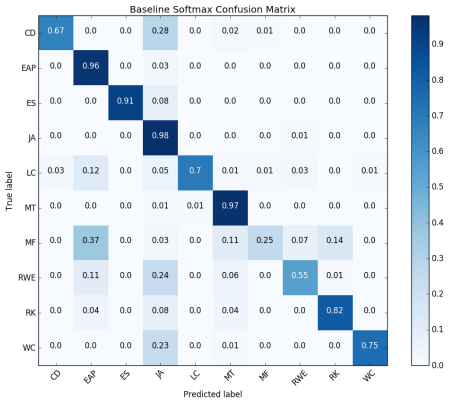

Figure 4: Confusion matrix for the baseline softmax classifier

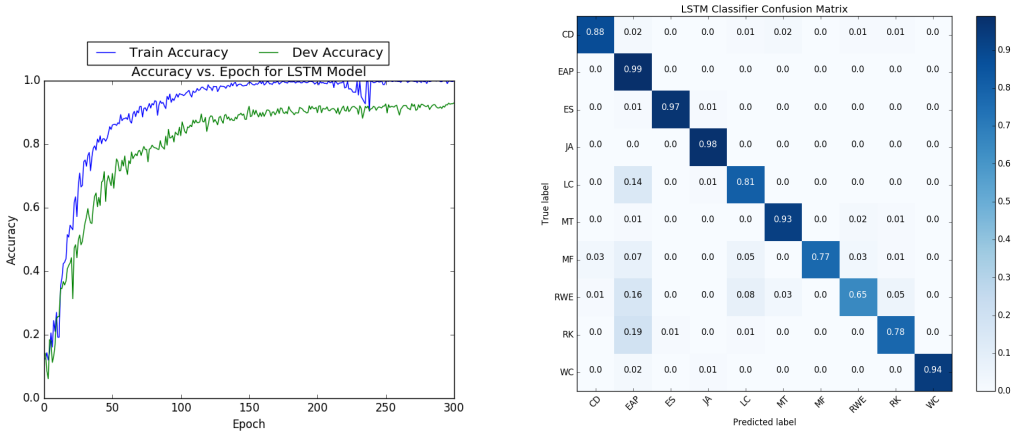

Figure 5: Training and development accuracies versus window size for the LSTM classifier (left) and the corresponding confusion matrix (right)

created a single generative model using data from all of the authors. For the latter, we also attempted to append a special token at the beginning of each sequence of words that indicated the author of the text. The idea was to give the model additional information about the inputted words, which could then be used to generate a more accurate next word. However, we found that even with hyperparameter tuning, this addition was subject to severe overfitting (training, development, and test perplexities of 112.2, 447.5, and 7,081.8, respectively). Thus, we left the special token out of our model's data setup. The hyperparameters used in these models are listed in Table 3. We used the gradient descent optimizer with gradient clipping to minimize our loss. We also used a dynamic learning rate to reduce the size of the model parameter updates as they approached the minimum. The results for the two sets of experiments are presented in Table 4. Note that for the single generative model experiments, the number of sentences in the training, development, and test sets are the same as for the per-author text generation experiments. Lastly, we trained a single generative model combining all of the data from all of the authors, achieving training, development, and test perplexities of 81.1, 112.1, and 135.5, respectively. Note that comparing the results from the total combined model to the individual models is not a fair comparison, as the total combined model had roughly 10 times the amount data to train on.

Author	Train	Dev	Test
Charles Darwin	38.5	156.0	164.8
Edgar Allan Poe	84.6	124.8	131.0
Edward Stratemeyer	59.8	88.0	109.0
Jacob Abbott	55.1	95.6	98.8
Lewis Carroll	55.3	127.1	168.6
Mark Twain	95.8	218.0	203.3
Michael Faraday	40.5	80.1	190.8
Ralph Waldo Emerson	72.3	592.0	470.5
Rudyard Kipling	109.1	213.0	284.7
Winston Churchill	77.4	154.5	145.3
All Authors Combined	87.8	202.6	201.6

Table 4: Train, development, and test perplexities for Text Generation

5 Discussion

5.1 Classification Baseline

The results for each classifier can be seen in Table 1. For both classifiers, the window size that maximized the development accuracy was around 700 to 800 words. This range of words maps to about 3-4 pages of text, which is a reasonable amount of information to predict the author of a work. Out of the two classifiers, GDA is more effective than NB for author classification, as the resulting test accuracy for GDA is almost double the test accuracy of NB. Overall, the results from the GDA classifier are notable, but do not surpass the results of the LSTM-based model.

5.2 Classification using Deep Learning

A test accuracy of almost 90% surpasses the results of Ge *et al.* and demonstrates that a well-tuned LSTM-model can achieve state-of-the-art authorship attribution results. Our LSTM model was able to predict the authors more accurately than both conventional machine learning models and a simple softmax classifier while using a smaller window size than both. Figure 5 also demonstrates that the LSTM model was able to discriminate between authors that the other models had trouble with (e.g. Michael Faraday in the case of NB and GDA and Charles Darwin in the case of the softmax classifier). We attribute this to the LSTM network's ability to mitigate vanishing gradient issues, enhancing its ability to use past words for author prediction.

5.3 Text Generation

Table 4 shows the training, development, and test perplexities for the per-author experiments. The text generation model for most authors scored a test perplexity between 100 and 200, which demonstrates the model to be relatively accurate in making the predictions. However, there were some outliers. In particular, *Ralph Waldo Emerson*'s text generation had a high test perplexity value of 470.5. Given *Ralph Waldo Emerson*'s transcendentalist style of writing poetry, it makes sense that our model had a lot of difficulty in generating the correct text for his body of work. Contrarily, *Jacob Abbott*, who wrote children's novels, had a simpler vocabulary that was more predictable, and this was evident in his model's low test perplexity value of 98.8.

The test perplexity for the model featuring all combined authors was 201.6, which is slightly higher than the average test perplexity of the per-author experiments (196.68). Given that the model is being trained on the myriad styles of the authors, a test perplexity value approximately equal to the average of the per-author experiments is expected. However, one of the limitations we encountered was the large amount of time needed to run each experiment, which led to smaller dataset sizes. The results from the total combined dataset in section 4.3 show that given more data and time, the model performs better and achieves a lower test perplexity.

6 Conclusions and Future Work

In this project, we implemented and compared multiple techniques for both authorship attribution and text generation. For author classification, we compared conventional machine learning techniques with an LSTM model, and showed the latter produced the highest test accuracy. For text generation, we implemented a per-author text generator and a mix-of-authors text generator. The per-author model performed well with some authors, such as Jacob Abbott and Edward Stratemeyer, and struggled with others like Ralph Waldo Emerson and Rudyard Kipling. The mix-of-authors task achieved a perplexity value approximately equal to that of the average of all the per-author tasks.

The primary limitation for this project and, subsequently, the main theme of the future work is the amount of time we had versus the amount of time the models needed to train. Thus, our future work falls into the following categories:

Larger Dataset: The Project Gutenberg dataset has over 100 authors, but training a model with 12,000 lines per author for 10 authors takes over 36 hours to complete. As we demonstrated in section 4.3, enlarging the dataset significantly improves the perplexity, but at a cost of longer model training times.

Parameter Optimization: The large amount of time needed to train the models resulted in a limited amount of parameter tuning, although we were able to do coarse-grain optimizations. While both models did quite well for their applied tasks, we believe there is still a potential to see improved performance with further parameter tuning.

Different Neural Network Models: As described in section 3.4, we used the LSTM-based neural network because of its robustness against the vanishing gradient problem, and also because of its ability to capture past information to enhance predictions. However, given more time, we would like to have compared the performance of the LSTM model to other additional models, such as the Gated Recurrent Unit or a Bi-Directional Recurrent Neural Network. We might also want to try a two-step network, such as a Convolutional Neural Network being fed into an LSTM unit.

Advanced Text Generation Per-Author: Our text generation model was based on the question of *given a window of text, can we predict the next word in the sequence?* We would like to extend this concept to develop an advanced network in which one or more authors are used as inputs to the model, and the model subsequently outputs sentences of text that are representative of the author(s) style. Such tasks have been performed with images, and would also be of interest in the NLP domain.

Acknowledgments

We would like to thank our mentor, Kevin Clark, for his guidance throughout the project, as well as Professor Christopher Manning and Richard Socher for making CS 224n an enjoyable and enlightening course. We would also like to thank Microsoft for giving us generous amounts of access to their Azure Cloud Computing platform.

Contributions

Greg: Implemented Conventional Machine Learning models for classification
Francisco: Implemented Deep Learning/LSTM models
Both: Data pre-preprocessing, poster, and project report

References

[1] K. Luyckx & W. Daelemans. Personae: a corpus for author and personality prediction from text, *in LREC,* 2008.

[2] F. Mosteller & D.L. Wallace. Inference and disputed authorship: The Federalist, *in Addison-Wesley,* 1964.

[3] F. Sebastiani. Machine learning in automated text categorization, *in ACM Computing Surveys,* 2002.

[4] F. Peng *et al.* Augmenting Naïve Bayes classifiers with statistical language models, *in Information Retrieval Journal,* 2004.

[5] Z. Ge *et al.* Authorship Attribution Using a Neural Network Language Model, *in Assoc. for the Advancement of Artificial Intelligence,* 2016.

[6] I. Sutskever *et al.* Generating Text with Recurrent Neural Networks, *in Proc. of Inter. Conf. on Machine Learning,* 2011.

[7] Gutenberg Dataset. `http://web.eecs.umich.edu/~lahiri/gutenberg_dataset.html`

[8] Name Database. `http://deron.meranda.us/data/census-derived-all-first.txt`

[9] GloVe Word Embeddings. `http://nlp.stanford.edu/projects/glove/`