EloquentRobot: A Tool for Automatic Poetry Generation

Jeffrey D. McGovern
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA
jmcgover@calpoly.edu

Gavin Scott
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA
goscott@calpoly.edu

ABSTRACT

Automatic content generation is a growing field with applications in creative writing and video-games. One aspect of lore and creative content is poetry, defined by features that may include, rhyme, repeated lines, word stress, meter, and parts of speech. Generating believable poetry can aid writers add more believability to their stories. To aid the task of content generation we built *EloquentRobot*, a Python-based tool that mines existing poems for structure and a corpus for content and procedurally generates poetry using the extracted features. By building a classifier for poetry structure, *EloquentRobot* was able to obtain a high accuracy for the classification of our generated poetry.

CCS Concepts

•Computing methodologies → Information extraction; Natural language generation; *Lexical semantics;* Language resources;

Keywords

poetry generation, natural language processing

1. INTRODUCTION

Automatic content generation is a growing field with applications in creative writing and video-games. Yearly competitions, like NaNoGenMo [1], challenge writers and programmers alike to generate a novel programmatically. Game worlds with the scale and explorability of *Elder Scrolls V: Skyrim*, the *Fallout* games, *Final Fantasy* games, and books with lore on the scale of *The Lord of the Rings* and *The Malazan Book of the Fallen* often have players spending hundreds of hours exploring the world, which can take an extremely large amount of creative manpower to create a believable world and its associated lore.

One aspect of lore and creative content is poetry. Poets have built a medium of natural language expression with defining features that may include, rhyme, repeated lines, word stress, meter, and parts of speech. Writers like J.R.R. Tolkien that build lore into their narratives often include poetry and songs as a part of it. Video-games set in the Middle Ages or that are of the fantasy genre often build similar volumes of lore that the player can discover throughout the game world. Generating believable poetry can aid these story writers and game developers quickly add more believability to their stories and video-games, enhancing the reader and player experience of the imaginary world they are experiencing.

In order to aid the task of content generation, we built *EloquentRobot*, a piece of software that can learn the relevant aspects of poetic structure from a set of example poems and generate new poetry in that form using content derived from a separate source of text. By building an algorithm that can learn from the structure of a set of poems and mine a corpus of content, we want to effectively zip up the two to create interesting poems. Existing poetry generation techniques [3] that use corpora typically use predefined templates of poetic structure, which we hope to achieve instead using data mining techniques.

EloquentRobot contributes the following:

- An algorithm to mine syllable and rhyme structure from a provided set of poems
- A set of features to mine from a corpus
- A method to combine these two mined data sources to generate poetry of the given style and using the provided content

This paper is organized as follows: Section 2 provides an overview of similar work, Section 3 discusses the data sources we used for poetry structure and corpora, Section 4 details the process *EloquentRobot* uses to mine data and generate poetry, Section 5 validates that *EloquentRobot* adheres to the correct poem structure, Section 6 concludes, Section 7 considers possible additions or alterations that may be interesting to investigate in the future, and Section B provides example poetry generated by *EloquentRobot*.

2. RELATED WORK

Previous work exists in the field of automatic poetry generation, often employing techniques similar to *EloquentRobot*, but differing in what they apply them to, while others focus on primarily Researchers in [21] developed a poetry generator that, like *EloquentRobot*, used two separate corpora: one for detecting form and another for generating content. They generated poems based on user-inputted topics,

BCB '16, October 02-05, 2016, Seattle, WA, USA
© 2016 ACM. ISBN 978-1-4503-4225-4/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2975167.2975205

and created the poem content by choosing words based on a word association network generated from a large corpus serving as a source of background knowledge. The poetry grammatical form was not generated, but pulled directly from randomly chosen examples of poetry, then the contents were replaced on a word-by-word basis based on POS tags. The primary focus of their work was on generating a cohesive content from a corpus of background knowledge, thus they ignored phonetic aspects of the poem structure (rhyme, syllable count, etc.).

Another content-focused poetry generation system [11] attempted to generate poems with an "emotional personalty" developed by reading texts and evaluating its emotional content. While this is outside the scope of *EloquentRobot*, their methods could justifiably be incorporated as future work into improving the generated poetry's content.

One work that somewhat resembles *EloquentRobot* is [4]. In it, researchers built an algorithm that could learn rhythm in poetry, i.e. they discover where word stresses occur on each line. Some of their difficulties mirrored our own, in that The Carnegie Mellon University Pronunciation Dictionary does not contain every word that shows up in a poem. Their focus was thus primarily on sonnets and iambic pentameter, which enforce stress as a structural requirement, thus allowing them to build a model agnostic of a pronunciation dictionary (but possibly augmented by one). Using the concept of a finite state transducer, they learn stress patterns of words by requiring that a sequence of words corresponding to a line of poetry in either sonnet or iambic pentameter fits the supposed rule of stress, using the Viterbi algorithm to find the most likely sequence of stress/no-stress. Generating poetry uses a trigram model of 10,000 lines of English love poems, selecting words that "realize" the desired stress. *EloquentRobot* differs in that it does not enforce prior domain knowledge of accepted poem types and focuses on generating poems using features extracted from any corpus, not just those of other poems.

3. DATA SOURCES

Generating new poetry using *EloquentRobot* requires two sources of data: poetry exhibiting the desired syllable and rhyme scheme to learn structure from and content to fill in the structure with.

3.1 Poetry Forms

PoetrySoup [16] contains poems categorized by type. By scraping the different sections of the website, e.g. http://www.poetrysoup.com/poems/haiku, we obtained a list of examples of different poem types. Reddit [19] also served as a source for example poems; it has a poetry section [14], where occasionally challenges [15] are put forth for certain poem types [6, 10]. There are also subreddits for different poem types, which we used to extract further examples of poem form.

We used PRAW [17], a Python API for accessing Reddit posts, to pull all of this data and add it to our poetry caches as well. It should be noted that all of the poems we scraped from online sources were user-submitted and did not undergo any evaluation before they were submitted, so many of them did not meet the strict definition of a haiku or a limerick. We judged this to be acceptable because the purpose of the tool is to generate poems that are similar in structure to a supplied corpus of forms, rather than following strict rules. To this end, when we evaluate the quality of our generated poetry we do so by comparing it to the poetry in the corpus, not the strict definition of the poetic form.

We focused primarily on building corpora of haikus [5] and limericks [9] from both Reddit and PoetrySoup because there were many more available examples of those than other poem types. We explored other sources as well but dismissed them because the organization of their poetry did not lend itself well to mining poems by form rather than other criteria (author, etc.). Bartleby[1] contains a wealth of poems, but were not sorted by style. Similarly, Project Gutenberg has a poetry bookshelf[2], but it had the same issues as Bartleby.

Below is a listing of the number of example poems we extracted, by form:

```
Limericks : 946
Haikus : 4488
```

In one of our evaluation methods we train a classifier on examples of poetry beyond limericks and haikus, to determine if our generated poetry matches closely enough to the source material to be classified correctly. We pulled couplets, quatrains, and cinquains from PoetrySoup, and unstructured free-form poetry from Reddit's `/r/OCPoetry` subreddit [12]. There were far fewer of these types available, and to keep the representation equal across types we limited each to only ninety-six poems.

3.2 Content

The content we used to fill in a learned structure came a four different sources, from which we mined *n*-gram-based features and pronunciation. The four content sources we used were the entire NLTK Brown Corpus, transcripts of Donald J. Trump's Presidential election campaign speeches, insults gathered from `/r/insults` [7], and curses gathered from `/r/TraditionalCurses` [22].

The Brown Corpus is a collection of English text compiled and published in the early 1960's by Brown University that represents texts from a multitude of genres. It was remarkable at the time because it was the first million word corpus published. Its texts are samples from writing in various genres including press reporting, editorials, reviews, religion, skills and hobbies, popular lore, biographies and memoirs, among others. It also includes general fiction short stories and novels as well as mystery and detective, science fiction, adventure, western, and romance and love stories. None of these categories factor into how *EloquentRobot* uses it to generate poetry, but future work may improve upon that.

Over the course of the Presidential election campaign, the website *What The Folly?!* [26] compiled the transcribed speeches of every Presidential candidate, including those of candidate Donald J. Trump. Using *BeautifulSoup* and *Python*, we scraped every article that claimed to be a transcript of one of Trump's speeches, extracting solely the speech text.

A website named Reddit [19] is a meeting place for people around the world that allows users to build their own subcommunity called a *subreddit*. Users can submit text-only posts or links to these communities that other Reddit users can upvote, downvote, and comment on. Communities are usually referred to by the text appended to the base Red-

[1]http://www.bartleby.com/verse/

[2]https://www.gutenberg.org/wiki/Poetry_(Bookshelf)

dit URL, e.g. `/r/canon`, for the community about Canon cameras.

Two of our corpora we built from two Reddit communities: `/r/insults` and `/r/TraditionalCurses`. Both of these contain primarily text posts where users post insulting content in the former and curses in the latter. Insults are generally stated and have no particular style — they are simply insulting, degrading, pejorative, or vulgar, among other things and sometimes all at once. Traditional curses follow a particular form that starts with the word "May", e.g.:

> *May the chocolate chips in your cookies always turn out to be raisins.*
>
> *May your tea be too hot when you receive it, and too cold by the time you remember it's there.*
>
> *May you be forced to repeat the most complex sentence that you have formed today because you didn't say it loud enough.*

Table 1 summarizes the distribution of data in our corpora.

4. METHODOLOGY

EloquentRobot builds a poem by mining a folder of poems that have a single structure, mining a corpus of text for features, and then combining them together. This section details these processes and overviews the tools and programming languages used.

4.1 Structure Mining

A poem structure is generated by mining a cache of poetry, extracting features from each and selecting features for a new poem based on the statistical distributions of these features throughout the poetry cache. The purpose of extracting features from a large number of poems rather than randomly choosing one poem to emulate was to come up with a structure that is more representative of the given type, while a single poem may have significantly parted from the form's strict definition. Also, should this approach be applied to a poem type with looser structure (such as free-form poetry), the resulting poem would have a form that is an amalgamation of the styles of each poem, rather than completely copying a single poem which could make it seem less "creative."

Each poem in a cache has both global and stanza-level features extracted from it. Global features are the number of stanzas as well as global rhyme schemes and line-repetition schemes (our program only used number of stanzas). For each stanza the number of lines and the rhyme scheme was extracted, as well as the following line-level features: part-of-speech (POS) tags for each word and the number of syllables in the line as a whole.

4.2 Content Feature Extraction

Features extracted from our content corpora are threefold: part-of-speech tags of words, n-grams of words and part-of-speech, and pronunciation for syllables counts and rhyming. Part-of-speech tagging allows us to enforce phrasing that makes sense in our poems. N-Grams allow us to generate believable, hopefully topical text based off of the content. Pronunciation allows us to use the content to build syllable and rhyme-structured poems.

4.2.1 Part-of-Speech Tagging

Part-of-Speech (POS) tags each word as it part of speech in a sentence. For the Brown Corpus, we used the supplied words with their part of speech. For the remaining corpora, which we custom built, we used the `pos_tag` utility built into NLTK.

4.2.2 N-Gram Extraction

N-Grams are the primary extraction of meaning for *EloquentRobot*. An *n-gram* is an n-length tuple consisting of word tokens that appear immediately next to each other in the text they originally came from and in the order they were originally in. Mining these features allows us to emulate a statistical Markov Chain process for natural language generation. Due to the nature of our pronunciation generation process for an n-gram, n-grams do not cross sentence boundaries. This process occurs for POS tags as well.

The approach to n-gram mining we took depended somewhat on the corpora we extracted from. The first step we needed to take was to tokenize the corpora, after which we are able to determine n-grams using NLTK's built in `ngrams` utility for a single n or `everygrams` utility which builds n-grams for a range of n.

Since the Brown Corpus was so massive, we only mined n-grams for $n = 2, \ldots, 5$. The Trump and `/r/insults` corpora were smaller, so we mined for $n = 2, \ldots, 6$. Lastly, the `/r/TraditionalCurses` corpus was the smallest of all the corpora, so we chose to mine for $n = 2, \ldots, 7$.

4.2.3 Syllables and Rhyming

Since the structure mined from our poem contained syllable and rhyme features for each line and the poem as a whole, we mined the syllables and pronunciation of the words in our corpora. Two tools developed by Carnegie Mellon University (CMU) helped us mine these features: the NLTK CMU Pronunciation Dictionary (CMUDict) and CMU's LOGIOS Lexicon Tool (LOGIOS). CMUDict is a dictionary of possible pronunciations for commonly used English words. LOGIOS uses letter-to-sound rules as a fallback if CMUDict does not contain the word, outputting the pronunciation using the same phoneme set as CMUDict.

The CMUDict is a lookup table for the pronunciations of 123,455 commonly used English words. It uses the set of phonemes detailed in Table 2, appending a "lexical stress marker" to each pronounced sound a digit, $0, 1, 2$ according to this rule:

- 0: No Stress
- 1: Primary Stress
- 2: Secondary Stress

We use lexical stress markers for two things: counting syllables and determining the operant rhyme phonemes. These pronunciations exist in the content database as well, for later debugging and reference. For smaller corpora, like `/r/insults` and `/r/TraditionalCurses`, we mined every possible pronunciation of the n-gram, while for especially the largest corpus, the Brown Corpus, we did not want to build a feature database that large.

Syllables in a word are simply those phonemes with any stress stress markers in them. By mining the CMUDict, we determined that The following phonemes always end up being syllables: AA, AE, AH, AO, AW, AX, AY, EH, ER, EY,

Table 1: Numbers showing the distribution of data in the corpora that *EloquentRobot* uses to generate poetry with. Documents are data sources for the Brown Corpus, number of speeches for the Trump corpus, and number of submissions for `/r/insults` and `/r/TraditionalCurses`. "First" pronunciation means that corpus took the first pronunciation of each n-gram provided, while "All" means we performed a cross product of all possible pronunciations of each gram in the n-gram.

Corpus	Documents	Words	n-grams	n-Range	Pronunciation
Brown	500	1,161,192	2,237,811	$2, \ldots, 5$	First
Trump	100	383,526	864,949	$2, \ldots, 6$	First
`/r/insults`	1,154	36,414	860,665	$2, \ldots, 7$	All
`/r/TraditionalCurses`	1,600	21,718	509,796	$2, \ldots, 6$	All

IH, IY, OW, OY, UH, and UW. Sentence boundary punctuation marks have no pronunciations, so n-grams containing them are thrown out, giving us only in-sentence n-grams.

Operant rhyme phonemes occur at the end of a word or n-gram, starting with the last stressed phoneme (either primary or secondary) and continuing to the end. This process captures masculine and feminine rhymes, but we do not distinguish between them. Since double rhymes consist of a primary stress followed by a secondary stress, we do not capture them.

4.3 Poem Generation

To create a new poem, the generator takes a form and a content database as input parameters. Looking first at the global features, it queues up the number of stanzas required by the form. Then it generates SQLite queries to produce each line in each stanza, using content n-grams pulled directly from the source database. Queries are written first by searching for a single n-gram that matches the following criteria:

- Has the same number of syllables as the line in the extracted structure
- If the line must have an end-rhyme that matches a previously generated line (the second and final line in a stanza with the rhyme scheme "aaba"), the query requires that the n-grams last word has a matching rhyme.
- If the line has a previously-unseen end-rhyme (the first and third lines in a stanza with the rhyme scheme "aaba"), the query requires that the rhyme of the n-gram is not a match for any previous rhyme.
- The last word of the n-gram has the same POS-tag as the last word of the line in the extracted structure.

While POS tags for each word in a line were included in poem forms, we only required generated poems to match POS on the last word of each line. This was decided because requiring a one-to-one match of POS was too restrictive a requirement, and even with a content source as large as the Brown Corpus, we were unable to generate a poem that met the requirements. However, we kept matching the last word's POS because we found that without that requirement the generated poetry would often be very nonsensical, using n-grams that ended mid-sentence, with articles like "the" or "of."

If an n-gram that met all the requirements could not be found in the content database the, query is split. A split query looks at first for two n-grams that combine to meet the line's requirements, with the sum of the syllables in each equalling the line's required syllable count and that the second n-gram matches the last-word POS tag and rhyme requirements. To maintain grammatical coherence, an additional requirement was added to the query that ensured that the last word of the first n-gram has the same POS tag as the first word of the second n-gram. When a pair of n-grams were combined, the first word of the second n-gram was thrown out to avoid duplication of the POS tag in the overlap (the syllable count was calculated without including the count of the thrown-out word).

This process continues up to an arbitrary number of splits (we used five), before the generator stopped and started the poem over from scratch. We very rarely saw a successful line generated with more than one split. Usually, if one split was not enough to meet the requirements, the problem was with the rhyme rather than the syllable count; if early in the poem a line was generated with a rare rhyme and a later line was required to match it, the content database would not contain any other n-grams that rhymed with the original word.

4.4 Implementation

EloquentRobot uses Python [18, 20, 23] and the Natural Language Tool-Kit (NLTK) library [2] to mine both poetry structure features and content features. Syllable counting uses CMUDict [25] at first and LOGIOS [8] as a fallback when CMUDict does not contain the word. Unfortunately, LOGIOS is not included in NLTK. Instead, we used *FreeTTS* [24], an open source text-to-speech library written in Java and with Maven, we assembled a single `.jar` file that returns a JSON representation of a list of pronunciations for the provided word, which our Python code calls. A SQLite[3] [13] database stores the content features to facilitate speedy querying for the n-grams that would make up a new poem. Using a SQL querying language took much of the burden of performing an efficient search of the data.

5. EVALUATION

The quality of a poem is difficult to quantify. Any metric would be too subjective; the best method would likely be to see if the content of the poem can be used to predict which database it was generated from, but because we are directly pulling n-grams from the content sources it is unlikely that a classifier would ever misclassify a poem based on it's content. For this reason, we did not attempt to evaluate the content of our generated poetry. That being said, we can evaluate how accurately our generated poetry matches the form of

[3]https://sqlite.org/

the poems that a generated poem was based on. To that end we built a training set from a combination of each of the poem types, where each entry consisted of the features of a poem, with the class equal to the poem type. We used a Naive Bayes classifier trained on this data set to attempt to classify newly generated poetry, and evaluated its accuracy. To test the classifier more completely we also included other poem types rather than just haikus and limericks (couplets, cinquains, and quatrains) in a second round of evaluation. To prevent a bias towards a poem type with more examples in the training data, in each test we truncated the list to include an equal number of examples for each type of poem, equal to the total number of whichever type had the least amount.

5.1 Limericks & Haikus

The first test used a classifier trained only on haikus and limericks, the poem forms with the largest poetry caches. The total training data set had 1892 poems (946 per type of poem). To evaluate the quality of the features used to train the classifier, we tested the training set against itself. The data was split in two, with two-thirds going to a smaller training set and the remaining third to a small test set. A Naive-Bayes classifier was trained on the small training data and used to classify the small test set, and the accuracy of these classifications was used to evaluate the quality of the classifier. For this test these classifications averaged an accuracy of 100% over one-hundred trials.

To test how accurately the forms of our generated poetry matched the source poems, we generated two-hundred poems of both types and used the same classifier, this time trained on the full training set, to measure how accurately it could predict the poem type. Like the test on the mined data, the classifier was able to correctly classify the type of poem with 100% accuracy in each test we ran.

5.2 All Poem Types

In a second round of testing we broadened the scope of the classifier to include all the other poem types we had pulled from PoetrySoup (couplets, cinquains, and quatrains). This training set had a total of only 480 poems, limited to 96 per poem type due to the limited availability of source material. The same two-thirds/one-third split of the training data yielded an accuracy of 94.82% over one-hundred trials. We generated two-hundred poems of each type again and ran them through the classifier after training it on the full training data, reaching an average accuracy of 93.8%.

5.3 Free-Form Poetry

We also extracted unstructured free-form poetry from Reddit's r/OCPoetry [12]. These poems have a much wider variety of structural forms, most unique to a single poem, but some with forms that match the other forms we had already extracted. We had planned to generate poems based on this corpus as well, but due to the length of the poems and the much larger average number of syllables per line the SQL queries became too computationally intensive. The algorithm would time-out generating long poems, only completing poems that matched the shortest examples from the corpus, preventing the generated set from representing the true distribution of features of the source data. However, we ran a third evaluation of the other five poem types, including the free-form poetry in the training data. This did not have a very significant effect on performance, with accuracies of 94.5% for the two-thirds/one-third baseline test and 93.3% when classifying generated poetry.

6. CONCLUSIONS

Automatic content generation is becoming increasingly relevant, due to the creative interests of writers and the creative need of large video-game developers. Whether an author of a story or game needs to get past writer's block, or simply needs to generate a large amount of interesting lore programmatically, there is a space for automatic poetry generation. For this task, we built *EloquentRobot*, a Python and NLTK-based tool that mines existing poems for structure and a corpus for content and procedurally generates poetry using the extracted features. Using haiku and limerick poetry from Reddit combined with corpora from different sources like the speeches of Donald J. Trump during his campaign, *EloquentRobot* is able to build cohesive poems. By building a classifier to classify poetry by its structure, *EloquentRobot* was able to obtain a high accuracy for the generation of its poem structure.

7. FUTURE WORK

In continue the project, we would increase the number of features extracted and replicated, possibly to include structural aspects like number of words per line, and line or phrase repetition. Then, to avoid over-constraining with features that are irrelevant to the provided poem type, we could identify which features of poem form (from a list of supported features) are the most important by filtering out features that are highly variant within the given poetry cache. If less-important features were identified and thrown out, the poem-generation process could discard their respective query constraints, resulting in a poem that is more true to the rules of the structure while filtering out less of the content database.

It would also be interesting to develop a way to improve the coherence of a poem, possibly by using a word-association map similar to other previous work in poetry generation [21]. Poem quality differed, depending on the corpus provided, primarily because *EloquentRobot* never considers topical factors, i.e. the topic coherence of the corpus defines how coherent the generated poem is with respect to a potential topic the poem may be about. This would reduce how nonsensical the generated poems tend to be, and could allow for poems to be generated not only with content from a specific corpus, but also focused on some given topic or theme. This would make a generation tool more customizable, increasing its usefulness.

To generally improve the tool, we'd also like to expand the number of types of poems we're testing with from just haikus and limericks, although the tool is already generalized enough that we would only need to find and input a large enough cache of a new type of poem to achieve this. Free-form poetry could be particularly interesting because it has no strict structural rules, but a definite and detectable style that would be interesting to try to emulate in the generated poems. Performance could also be increased if a method of backtracking was implemented; the algorithm currently gives up and restarts the poem if a line cannot be generated with the given constraints, but for a longer poem it could save time to identify if going back and replacing earlier lines would be quicker than starting from scratch.

8. REFERENCES

[1] NaNoGenMo 2016: National Novel Generation Month. https://github.com/NaNoGenMo/2016.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. "O'Reilly Media, Inc.", 2009.

[3] Simon Colton, Jacob Goodwin, and Tony Veale. Full-face poetry generation. In Mary Lou Maher, Kristian J. Hammond, Alison Pease, Rafael Pérez y Pérez, Dan Ventura, and Geraint A. Wiggins, editors, *Proceedings of the Third International Conference on Computational Creativity, Dublin, Ireland, May 30 - June 1, 2012.*, pages 95–102. computationalcreativity.net, 2012.

[4] Erica Greene, Tugba Bodrumlu, and Kevin Knight. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 524–533. Association for Computational Linguistics, 2010.

[5] Haiku. https://reddit.com/r/haiku.

[6] Haiku Challenge. https://www.reddit.com/r/Poetry/comments/1ykx7r/cc_haiku_challenge_and_a_poll_click_here/.

[7] Insults. https://reddit.com/r/insults.

[8] K-F Lee, H-W Hon, and Raj Reddy. An overview of the sphinx speech recognition system. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(1):35–45, 1990.

[9] Limerick. https://reddit.com/r/limerick.

[10] Limerick Challenge. https://www.reddit.com/r/Poetry/comments/2ko2b7/cc_challenge_write_a_limerick/.

[11] Joanna Misztal and Bipin Indurkhya. Poetry generation system with an emotional personality. In *Proceedings of 5th International Conference on Computational Creativity, ICCC*, 2014.

[12] Original Poems. https://reddit.com/r/OCPoetry.

[13] Mike Owens and Grant Allen. *SQLite*. Springer, 2010.

[14] Poetry. https://www.reddit.com/r/Poetry.

[15] Poetry Challenge. https://www.reddit.com/r/Poetry/search?q=%5Bcc%5D&restrict_sr=on&sort=relevance&t=all.

[16] PoetrySoup. http://www.poetrysoup.com/.

[17] PRAW: The Python Reddit API Wrapper. https://praw.readthedocs.io/en/latest/.

[18] Python. https://www.python.org/.

[19] Reddit. https://reddit.com/.

[20] Guido Rossum. Python reference manual. 1995.

[21] Jukka Toivanen, Hannu Toivonen, Alessandro Valitutti, Oskar Gross, et al. Corpus-based generation of content and form in poetry. In *Proceedings of the Third International Conference on Computational Creativity*, 2012.

[22] TraditionalCurses. https://reddit.com/r/TraditionalCurses.

[23] Guido van Rossum and Jelke de Boer. Interactively testing remote servers using the python programming language. *CWi Quarterly*, 4(4):283–303, 1991.

[24] Willie Walker, Paul Lamere, and Philip Kwok. Freetts 1.2: A speech synthesizer written entirely in the java programming language, 2010.

[25] R Weide. The CMU pronunciation dictionary, release 0.6, 1998.

[26] What The Folly?! Transcripts. http://whatthefolly.com/transcripts.

APPENDIX

A. EXAMPLE POEMS

Below are select poems from each corpus. There are two poetry type examples: haiku and limerick. Poetry type inference will be left to (and should be obvious to) the reader. One disclaimer we posit is that since certain corpora come from the Internet, some parts of the resulting poems are inappropriate in ways that our algorithms cannot detect, but we include them to exhibit the capabilities of *EloquentRobot* to dynamically extract lexical structure.

A.1 Brown Corpus

In the lake placid area
was typhoid and malaria
the life eternal
to an external
was beset by hysteria

of hospitals lifted the ban
can love eisenhower the man
early romantic
look like gigantic
criticized the president's plan

contain a design
entitled to any such
shadow on the grass

In considerable detail
petitions asking for a jail
one cataclysmic
of organismic
equipped with a full beavertail

since the election
nations confirm that the cold
it is almost time

subsequent disinclination
grew out of a combination
serves only its own
she is even prone
for the rehabilitation

poets longing for his homeland
down by buying good secondhand
appearing in such
supplied him with dutch
is surrounded by a wasteland

in a rational lexicon
historical phenomenon
was wearing a brown
years he wore handmedown
promoted to be liaison

A.2 Trump Transcripts

Sarah Palin has been from day
cases than they're making today
trillions in the long
that it must be strong
Second Amendment by the way

going to build a great border
get approval to build the wall
American people
get from people
We're bringing your education

stay home and follow traditions
stay home and follow traditions
Thank you very much
that I won by such
stay home and follow traditions

some people say We love
Today were giving a big
to speak from the heart

would magically join the world
the seventh wonder of the world
But that's like a good
because they're so good
stupidly all over the world

most basic duty
be extremely dishonest
taxing and her tax

where there's value in exchange
going to deliver real change
So that's pretty good
Bergdahl a no good
ballistic missiles with a range

the morals of an alley cat
that of a leprous desert rat
Your mums like a shit
here is subreddit
fat pig rotten at your core fat

me money okay
Honor and many many
I just signed a lease

their people back when we order
down to Texas to the border
has had so many
I dont give any
you want me said our border

A.3 Insults

There were many times
of sexual violence
the hottest phrases

worthless bag of filth
just like to kill innocent
I just lost my job

sex for the first time
and I aim to keep it nice
of hepatitis

I've found the biggest thing
And found your dad's wedding ring
jump in you get wet
I fell off my pet
love me love you xoxo mong ding

private prep school and a nice house
to spit Stop showing me your spouse
Jews and fucking fuck
your stupidasfuck
and right disinfected shithouse

A.4 Traditional Curses

walk through a doorway
you step on a Lego piece
when you bend over

always hit the edge
to get that bit of popcorn
use gojo pumice

forget why you entered a room
scratches on your arm and assume
May you receive nude
you be so pursued
when your parents enter the room

lever on your computer chair
rub the shampoo into your hair
pee before a long
always buy the wrong
on the edge of every stair

of seconds before the new year
inadequate in the career
never have the right
May your least favourite
of water never leave your ear

the beginning of your toilet
to cough just fall in the toilet
always use the wrong
two fat men on long
second guessing if the toilet

B. PHONEMES

Table 2 describes the phoneme set that the Carnegie Mellon University Pronunciation Dictionary uses for the pronunciation of words. The LOGIOS Lexicon Tool uses the same phoneme set. Not show in Table 2 are lexical stress markers, used to determine which syllables are primary, secondary, or not stressed at all.

Table 2: The CMU Pronunciation Dictionary phoneme set with examples.

Phoneme	Example	Translation
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY
EH	Ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY
L	lee	L IY
M	me	M IY
N	knee	N IY
NG	ping	P IH NG
OW	oat	OW T
OY	toy	T OY
P	pee	P IY
R	read	R IY D
S	sea	S IY
SH	she	SH IY
T	tea	T IY
TH	theta	TH EY T AH
UH	hood	HH UH D
UW	two	T UW
V	vee	V IY
W	we	W IY
Y	yield	Y IY L D
Z	zee	Z IY
ZH	seizure	S IY ZH ER