CS 221 Group Project

Atticus Geiger, Fangzhou Liu, Amit Gamane

November 17, 2016

Our project aims to generate alternative rhyming couplets for Shakespeare sonnets. The input would be the first 12 lines of a Shakespeare sonnet and the output would be a rhyming couplet to complete the sonnet. Ideally, the produced couplet will have the tone of Shakespeare and make sense given the previous 12 lines. To accomplish this task, we plan to use a training set of 100 of Shakespeare's 152 14-line sonnets. Our preprocessing phase will take place in two parts: as an initial approach, we will use an unsupervised word embeddings approach. In essence, this applies a parameterized function on a given word to produce a high-dimensional vector. Applying a shallow neural model on these vectors will allow us to create a model. We can then use this to later help determine costs in the search algorithm. Second, we will extract the syntactic structure from the 100 sonnets and select certain phrase structures to generate random syntax templates for our output couplets. Once a syntax template is chosen, a search problem is modeled as seen below using the syntax template and word clusters; the syntax template defines the actions and states of the search model, while the word clusters define the cost function.

In this progress report, we summarize our approach to generating grammatical couplets with 10 syllables in each line. Additionally, we cover some ground on our initial learning model to build up our cost function. Future work includes imitating the rhyme and iambic pentameter of Shakespeare's couplets, as well as taking into consideration the meaning of the preceding 12 lines and exploring other possible unsupervised learning methods.

Syntax Template Selection

During preprocessing, we generate a syntax template comprising of part-of-speech category labels corresponding to each individual word to be included in the couplet. An example template is as follows:

DT NN POS JJ VB V TO VB : DT NN , NNS VNP .

Each token represents the part-of-speech category corresponding to each word in the couplet. For each couplet, we generate a new syntax template by randomly combining a series of syntactic rules, such as the following:

S $\rightarrow$ NP VP : NP , NP VP .
VP $\rightarrow$ V ADJP
NP $\rightarrow$ NP CC NP

We extracted these rules by parsing the couplets of Shakespeares first 100 sonnets with the Stanford Dependencies parser. The parser output syntactic tree representations of each couplet, where every part-of-speech category of a word is a terminal node (e.g. V for verb), with some phrase node (e.g. VP for Verb Phrase) as a parent. Given the syntactic tree representations of each couplet generated by the parser, we extracted the following sets of phrase structure rules for the non-terminal nodes:

S rules (Sentence; that is, full couplets)
VP rules (Verb Phrases)
NP rules (Noun Phrases)
ADJP rules (Adjective Phrases).

Where S rules are comprised of VPs, NPs and ADJPs. Notice that the rules allow for recursion, where NP, for example, can correspond to NP CC NP.

In this early phase of the project, we limited ourselves to 1 rule for S, selected for brevity to better accommodate syllable count restrictions. For every new couplet that we generate, we use this rule for S and randomly select NP, VP and ADJP rules from the list to output the syntax template that consists solely of part-of-speech categories for individual words.

Currently, the VP, NP, ADJP and S rules have been selected by hand, with the intention of limiting the length of each subphrase to avoid exceeding the syllable count per line. Possible future work includes further analyzing the syntax of the first 100 couplets to extract their most common phrase structures for our use. In addition, we aim to aggregate the average syllable count for each phrase-type to so that the syllable count may be included in the cost function as every action (word) is taken, rather than including this cost only at the last state.

Search Model

Given a sequence of parts of speech $X = (X_1, X_2, ..., X_n)$, let the following search problem be defined. Let $X_{n+1} =$ "END_TOKEN" indicate the end of the line. Let f be a function that takes in a part of speech and returns the set of all available words that are that part of speech. Let g be a function that takes in a word and returns the number of syllables. Let h be a function such that, $h(Y) = \frac{1}{|f(Y)|} \cdot \Sigma_{word \in f(Y)} g(word)$, so h takes in a part of speech Y and returns the average number of syllables contained in a word that is that part of speech. Let a state be defined by the number of syllables that have been chosen so far, and the next part of speech that is to be chosen.
Let the start state be defined to be $S_{start} = (0, X_1)$.
Let $Actions((k, X_i)) = f(X_i)$.
Let $Cost((k, X_i), a) = 10000$ if $i == n$ and $k + g(a)! = 10$ else $|10 - (k + \Sigma_{j=i}^{n} h(X_j))|$.

Let $Succ((k, X_i), a) = (k + g(a), X_{i+1})$.
Let $IsEnd((k, X_i)) = [i == n + 1]$.
Uniform Cost Search is used to solve for the lowest cost path, and the actions that form the lowest cost path are the line. The cost of a word is to guide the algorithm to find a line consisting of 10 syllables.

Here is a small example input-output:
Input:
Let $X = (NP, V, N)$.
Let $f(NP) = \{I\}$, $f(V) = \{eat\}$, $f(N) = \{food\}$.
Then $g(x) = 1$ for all x.
Then $h(Y) = 1$ for all Y.
Output: I eat food

The functions f, g, and h were created using the actual Shakespeare Sonnets set aside for training. Using the nltk part of speech tagger and the cmu pronouncing dictionary, every word in the training set was assigned a part of speech and a number of syllables. If the cmu pronouncing dictionary did not contain some of the words in a line, then because each line of a Shakespearean sonnet contains 10 syllables, the syllables that each unknown word contains is estimated by dividing the number of remaining syllables in the line between the unknown words weighting on word length. For example, if a line is "My name is Atticus Geiger, he said" and the cmu dictionary did not recognized "Atticus" and "Geiger", the remaining 5 syllables would be divided as such Atticus:3 and Geiger:2, because Atticus is just barely longer than Geiger. Finally, once all words are assigned a number of syllables and part of speech, h is calculated using f and g because $h(Y) = \frac{1}{|f(Y)|} \cdot \Sigma_{word \in f(Y)} g(word)$.

Results

A shop had vile lace to lace a forget
A shop had vile lace to lace a shop suns

Learning Model

In the context of the search problem mentioned, we need a way to come up with a objectively way to cost our choices during the search process. As such, we decided to take an unsupervised learning approach as it would be hard (and perhaps even less useful) to go about a form of supervised learning.

As such, the scope of this section allows us to consider some variations of unsupervised learning models. For a start, we decided to go with a word embedding model and applying learning on the generated high-dimensional vector. More specifically, words that are semantically closer will generate a higher score than otherwise. As such, we decided to use word2vec, a shallow neural learning model developed by Mikolov et al. word2vec employs a continuous bag-of-words model (CBOW) and a skip-gram model. The difference in the 2 lies in the fact that CBOW is good at prediciting words from a given source corpus while the skip-gram does the opposite and looks to predict more source words given a specific input word.

In terms of learning, word2vec itself employs a shallow(bi-layer) neural network. The input would be a given text corpus, specificially, a list of sentences to be intepreted and the output layer gives us a set of a high dimensional feature vectors that can be described as the vocabulary of the corpus. This vectors can then be fed into a another model or can be queried for similarity between the word-vector representations. This similarity evalution is done using cosine similarity.

Approach + Results

To prepare the corpus (sonnets) to be fed into word2vec, some preprocessing was required. This involved gathering all the sonnets and performing chunking via a stoplist. This would remove common connectors used in the language and would allow the results later to be more meaningful. This was done using a NLTK provided stoplist. After this was done, punctuation was stripped from the corpus to keep intact only the semantic meaning of the words. With the corpus now prepared, it was fed into word2vec to generate a model.

In order to guage the rough similarity accuracy of the model, some test examples were used and yielded the following scores:

Perfect example: model.similarity('eye', 'eye') $\rightarrow$ 1.0000

We then came up with words that used in Shakespearean literature and construct pairs of positively and negatively-correlated words.

- (+) ('eye', 'sight') $\rightarrow$ 0.08665
- (+) ('bright', 'fire') $\rightarrow$ 0.09805
- (+) ('beauty', 'love') $\rightarrow$ 0.08045
- (-) ('shop', 'vile') $\rightarrow$ 0.02987
- (-) ('work', 'lovers') $\rightarrow$ -0.13517
- (-)('crystal', 'shop') $\rightarrow$ -0.12768

Hence, it can be observed that positively related words tend to score higher. However, the limitation of the size of corpus would mean that more extensive testing needs to be done. Ultimately, this model needs refinement to become a performant cost function for the search problems, along with the need for other methods to be explored.