POS-Gram: Generating Poetry Couplets Based On A Source Document

Miller Tinkerhess

ECCS 595 Project 1

3/25/2011

Abstract

The goal of this project is to be able to feed an example of prose text into an algorithm, and then generate a poem from the text. One common approach to generating text based on a source document is N-grams. In this paper I present POS-gram, a modification to the N-gram algorithm that takes into account the parts of speech of the previous generated words in addition to the recently generated words themselves. I explore approaches to rhyming, syllabic emphasis, line relatedness, and flexibility, in order for the generated text to resemble poetry. The complete couplet-generation algorithm uses POS-gram to randomly generate couplets, and then performs checks and modifications on the couplets in order to satisfy constraints; if the constraints cannot be satisfied, the couplet is discarded and a new one is generated. In choosing an appropriate value for *n* under the N-gram model, a tradeoff has to be made between syntactic and semantic cohesion for large values of *n*, against the flexibility of a large search space for small values of *n*; I show that POS-gram is a feasible alternative to N-grams that exhibits some of the positive features of both low-valued and large-valued N-grams.

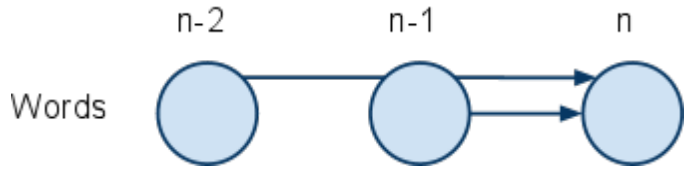
Implementation and Variations

I implemented POS-gram in Python, making use of the Natural Language Toolkit (NLTK) library for access to Wordnet and part of speech tagging, and the CMU Pronouncing Dictionary to determine the number of syllables in each word, the emphasis of each syllable, and which words rhyme with each other. Throughout the development process I used the first chapter of Lewis Carroll's *Alice in Wonderland* as a source document, which is 2,161 words of fanciful, playfully-punctuated prose. Anecdotally, I applied the algorithm to other texts to make sure I wasn't designing the algorithm to match the source text too closely, but I do not present results from other source documents here. I compare word-based 2-gram and 3-gram approaches with various implementations of POS-gram.

The first step I use under any approach is to tokenize the source document. I use NLTK's default tokenizer and part-of-speech tagger, along with custom heuristics to avoid various pitfalls I encountered. Sentences are separated by periods, questions marks, exclamation points, colons, semicolons, and parenthesis; dummy 'BEGIN' and 'END' tag/word pairs are added to the beginning and end of each sentence. Punctuation tokens receive their own tag/word pairs, separate from words. In some

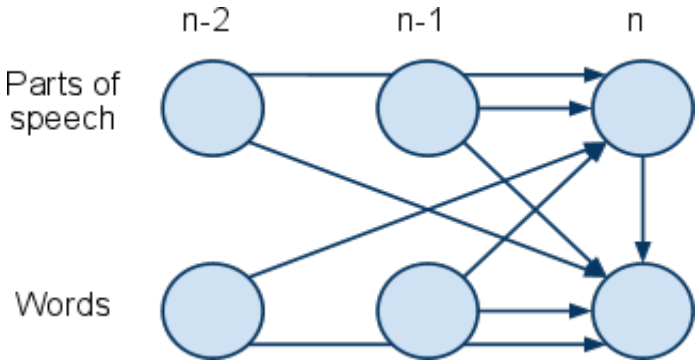
cases there are bugs in the tokenizing algorithm such that a token contains both a word and some punctuation; this is unfortunate but rare enough that it is not a major problem.

A common technique for generating new text in the style of a source document is N-grams. In this approach, for each set of *n* words that occurs in sequence in the source document, the occurrence of each word that follows that set is counted to determine a probability distribution. When generating text, the *n* most recently generated words are considered, and the next word is chosen from that group's probability distribution. For larger values of *n*, the generated text is more likely to resemble syntactically correct text, and is also more likely to quote segments of the source document verbatim. Smaller values of *n* allow for more flexibility because the space of texts that may be generated is higher.



Dependencies between words under a 2-gram model.

POS-gram first tags the source document with parts of speech for each word, and then forms a Markov chain where new part of speech / word pairs are generated based on the previously occurring words and part of speech tags. First, a new part of speech tag is chosen based on the previous two parts of speech and the previous two words; then a new word is chosen based on the new part of speech, the previous two parts of speech, and the previous two words. Compared to N-grams, this approach has some of the benefits of both small and large values of *n*: by taking part of speech tags into account, the result is more likely to by syntactically correct while only looking backward a small amount, so that the generated text is less likely to quote segments of the source document verbatim, and a larger space of texts may be generated.



Dependencies between parts of speech and words under POS-gram. At each step, first POS[n] is generated, then word[n].

In order to generate poetry, constraints must be made on the generated text. I have chosen to generate text in pentameter form, with rhyming pairs of lines of ten syllables each. The first way to satisfy the syllable-count constraint is to generate sentences of more than ten syllables, and remove words form the beginning of the sentence until it reaches ten syllables; if the line cannot be made to fit in ten syllables, it is discarded. This has the advantage of discarding very few lines; most lines can be made to fit into 10 syllables. However, the beginning of each line is often nonsensical, such as a conjunction or preposition that doesn't make sense in the context of the whole line. I explore two ways to mitigate this problem. First, I discard lines in which the first word of the line never appears as the first word of a sentence in the source document. This eliminates the most egregious offenders, but the beginnings of the lines that remain are often drawn from probability distributions relating to the middle of sentences, not the beginnings. Second, I only accept lines that have ten syllables, without removing any words from the generated sentence. This means that the beginning of each line is generated according to the probability distributions of the beginnings of sentences (like the end of each line), which means the the whole line tends to feel more like a complete thought.

The final constraint that may be applied to a single line is syllabic emphasis. I test three approaches to emphasis constraints: no constraints, each line must be in iambic pentameter, or each line muse be in either iambic or trochaic pentameter. Iambic pentameter restricts the line to the pattern weak-strong-weak-strong, etc; trochaic pentameter restricts the line to the pattern strong-weak-strong-weak, etc. The CMU Pronunciation Dictionary includes emphasis information for each syllable in the range 0-2. To satisfy emphasis constraints, the emphasis value of each strong syllable must be of greater or equal value than the value of the previous syllable, and the value of each weak syllable must be less than or equal to the value of the previous syllable.

A couplet is a pair of two lines with some meaningful connection between them. One possible such connection is rhyme. In order for two lines to rhyme, the last words of the two lines must rhyme with each other. Two words rhyme with each other if the pronunciations of both words, from its last emphasized syllable to the end of the word, are identical. This occurs by chance rarely enough that my normal approach of discarding pairs of lines that don't fulfill a constraint would make the problem intractable. Instead, once two lines are generated, I consider the possible assignments of each line's final word in under a more permissive version Markov chain used to generate the final word. Pairs of assignments that rhyme with each other are kept; others are discarded. If no pairs of assignments are kept, the couplet is discarded; otherwise, a rhyming assignment is chosen with probability proportional to the product of the probabilities of choosing each assignment independently.

Another way two lines in a couplet may be related is through semantic relationships. POS-gram generates lines without taking into account semantic meaning. Hopefully, the generated lines will contain some meaningful essence of the source

document; however, that meaning is not represented explicitly in the algorithm. On the other hand, semantic information about the individual words can be accessed through Wordnet. Assuming that the semantic meaning of a line in a couplet is some function of the semantic meaning of its constituent words, two lines may be semantically compared to each other by semantically comparing their words. Past work has explored the use of semantic relationships between words senses in Wordnet to generate Haikus [Netzer et al., 2009].

I use synonym, hypernym and hyponym relationships to determine whether or not two words are related to each other. Each word in the first line is compared against each word in the second line. For each word, consider all the synsets it belongs to. Additionally consider all the hypernym and hyponyms of those synsets. If the second word belongs to any of those synsets, the words are related. While this heuristic is not always accurate, it does often find pairs of words that have some obvious connection. By counting the occurrence of these relationships between two lines, the relatedness of the lines may be scored. To compensate for the over-optimism of this similarity heuristic, lines may be considered similar if their similarity measure is above some threshold. The higher the threshold, the more likely that one or more of the relationships is not a false positive; i.e. a true relationship in Wordnet but not meaningful to a reader, either because of the context of the words of the obscurity of their relationship. I have found that using a threshold of 2 rends to generate interesting pairs of lines without restricting the search space so much as to make the problem intractable.

Results

I ran experiments on 8 variants of the POS-gram algorithm, as well words-only 2-grams and 3-grams. For each variant, I generated 1000 couplets in order to amortize the variations inherent in the random-search approach. To measure approximate performance of each variant, I measured the number of couplets that had to be generated in order to produce the 1000 satisfactory couplets. The results show that POS-gram is able to take advantage of the parts of speech of the two previous words to generate text; that runtime grows exponentially as you add more constraints to the algorithm; and that POS-grams are more likely to produce sentences that can rhyme with each other.

Variants Tested

	Markov Algorithm Used	Rhymes	Require first word to be a real first word	Remove first word until 10 syllables reached	Only accept line if it's already 10 syllables	Require iambic Emphasis	Require iambic or trochaic emphasis	Require WN-relatedness >= 2
2-grams	2-Grams	X		X				
3-grams	3-Grams	X		X				
Blank Verse	POS-Grams			X				
Default Options	POS-Grams	X		X				
First Words	POS-Grams	X	X	X				
Full Sentences	POS-Grams	X			X			
Iambic Emphasis	POS-Grams	X		X		X		
Iambic or Trochaic Emphasis	POS-Grams	X		X			X	
Sentences and Wordnet No Rhymes	POS-Grams				X			X
Wordnet Relatedness	POS-Grams	X		X				X

Features of each variant tested.

2-Grams and 3-Grams

A regular Markov chain, not taking parts of speech into account.

Blank Verse

Lines are not required to rhyme.

Default Options

Lines are generated using the POS-gram algorithm, and required to rhyme. In order to fit each line to 10 syllables, leading words are removed until the line is the right length.

First Words

Trimmed lines are only accepted if the first word in the resulting line appears as the first word of some sentence in the corpus.

Full Sentences

Generated lines are only accepted if they are already 10 syllables, without any trimming.

Iambic Emphasis

Trimmed lines are only accepted if their syllables' emphases are in iambic pentameter.

Iambic of Trochaic Emphasis

Trimmed lines are only accepted if their syllables' emphases are in iambic or trochaic pentameter.

Sentences and Wordnet, No Rhymes

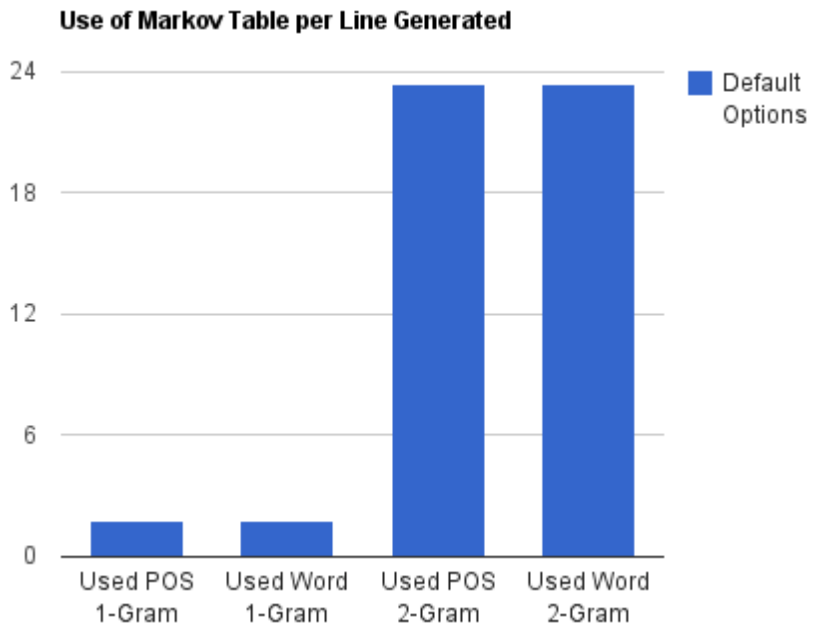
Generated lines are only accepted if they are 10 syllables long without trimming. A couplet is only accepted if its lines have 2 or more pairs of words whose hypernym /

hyponym distance is less than or equal to 1.

Wordnet Relatedness

A couplet is only accepted if its lines have 2 or more pairs of words whose hypernym / hyponym distance is less than or equal to 1.

Feasibility



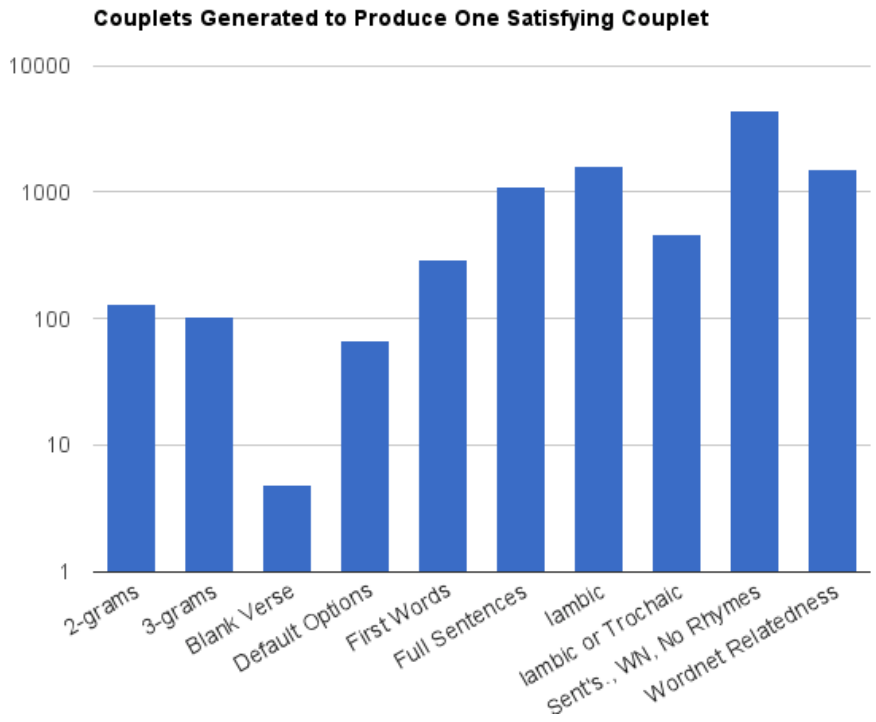
These results show the feasibility of adding part of speech information to a Markov chain. That is, this shows that including part of speech information in the Markov tables does not overly restrict the space of possible results; given a partially-generated chain produced by POS-gram, the last two words and parts of speech are very likely to be valid keys for the Markov tables, so the chain may continue to be generated using all four recently-generated components (two words and two parts of speech).

If, on the other hand, using part of speech data overly restricted the result space, you would find that your Markov tables have no entry that matches all four recently generated components, and you would have to revert to the Markov table that only looks at the two most recent components instead of four.

This data very closely resembles the data collected for 2-grams and 3-grams (not reported here), which indicates that including parts of speech in the Markov tables does not restrict the result space due to invalid keys any more than N-grams do; therefore this is a feasible alternative to N-grams.

Runtime

	Couplets Generated per Satisfactory Couplet
2-grams	129.89
3-grams	101.97
Blank Verse	4.82
Default Options	66.45
First Words	293.7
Full Sentences	1090.43
Iambic	1586.98
Iambic or Trochaic	461.67
Sent's., WN, No Rhymes	4380.41
Wordnet Relatedness	1494.6



Because the time required to generate a single line is relatively constant, the number of randomly-generated couplets required to produce a single satisfactory couplet is a good indicator of the algorithm's performance. This data shows that as you add more restrictions to the algorithm, its runtime grows exponentially. Beginning with blank verse, adding the requirement that lines rhyme with each other increases runtime from 4.82 to 66.45, or 1,379%. Adding the further first-word restriction increases runtime to 293.7, or another 442%. Requiring that the sentence be 10 syllables long without any trimming increases runtime to 1090, or another 371%.

Moving from the default options to the requirement that lines fall into either iambic or trochaic pentameter increases runtime from 66.45 to 461.67, or 695%. Restricting lines to only iambic pentameter further increases runtime to 1,586.98, or another 344%.

Moving from the default options to requiring Wordnet-relatedness increses runtime from 66.45 to 1494.6, or 2,249%. Attempting to require Wordnet-relatedness on rhyming, whole sentences proved intractable for generating 1,000 couplets, so I removed the rhyming restriction to make the problem feasible. Still, requiring both Wordnet-relatedness and full sentences even without the rhyming requirement increased runtime to 4380.41, or 293% as much as Wordnet-relatedness alone.

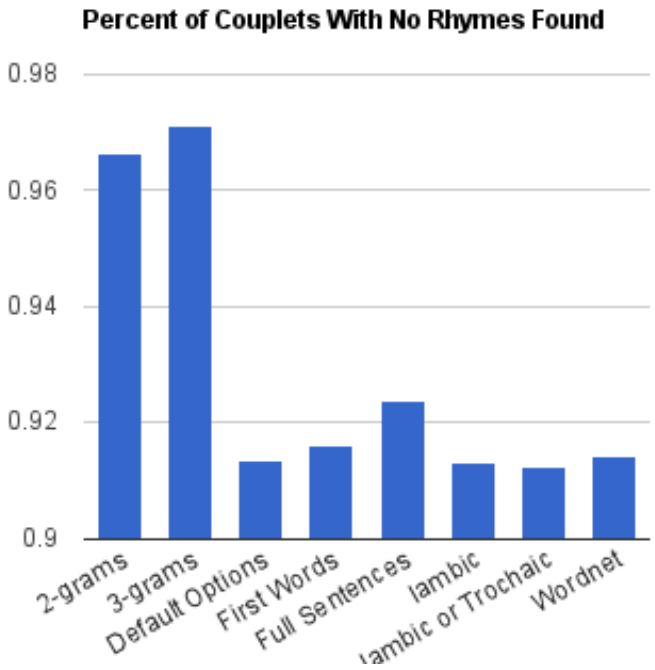
The performance of 2-grams and 3-grams is comparable to performance of POS-gram. In fact, under the rhyming constraint POS-gram slightly outperforms N-grams because it is more likely to produce lines that can rhyme with each other.

The two constraints that most impact runtime are rhyming (at 1,379%) and Wordnet-relatedness (at 2,249%). Therefore, attempts to improve performance of POS-

gram should focus on those areas. For example, other types of rhymes such as slant rhymes could be admitted in addition to the strong rhymes currently supported, or a less restrictive definition of semantic relatedness could be used.

Rhymes Found

	No Rhymes Found
2-grams	0.966125182847
3-grams	0.971167990585
Default Options	0.913468773514
First Words	0.915934627171
Full Sentences	0.923580605816
Iambic	0.912903754301
Iambic or Trochaic	0.912383304092
Wordnet Relatedness	0.914244613944



This data shows the percentage of couplets generated under each approach for which no rhyming pair of final words was found. This may be due to two factors: the method by which the lines are generated, and therefore the probability that a pair of generated lines may be manipulated into rhyming; and the method by which two lines are manipulated into rhyming.

The general approach to manipulating a pair of lines into rhyming is as follows: the last word of each line is removed. The remaining final couple of words (and under POS-gram, parts of speech) are considered, and the set of all words that may end each line is found. The set of possible ending words for each line is considered, and pairs of items from both sets are gathered in which the pair rhymes. If no rhymes are found, the couplet is discarded; otherwise, a rhyming pair is selected and assigned to the couplet.

In order to expand the possible rhymes, the less restrictive Markov tables are used to select candidate ending words; e.g., the 1-gram table under the N-gram approach, or the table mapping POS(n-1), word(n-1), and POS(n) to word(n) under POS-gram. Intuitively, POS-gram applies more restrictions to the set of candatite final words, and should therefore fail at finding rhyming pairs more often than N-grams. However, the data shows the opposite result, which indicates that there is something inherent to the kind of sentence produced by POS-gram that makes the ends of its lines more likely

to rhyme than N-gram-produced lines. Perhaps POS-gram produces ends of sentences that are more similar to "real" ends of sentences than N-grams, and therefore are more likely to contain similar kinds of words, which are more likely to rhyme with each other? These observed differences could also be the result of characteristics of the source document—given a different source document, perhaps N-grams would find rhyming pairs more often than POS-gram.

Example Generated Couplets

Here are examples of couplets that are generated by each approach. These are the first eight couplets generated by each algorithm; I didn't do any hand-picking of examples.

These examples reveal some of the interesting properties of the variants that are difficult to quantify. Despite their subjectiveness, these properties are more important to a reader than any statistics that may be gathered, and are the true goals of this project. If this project is a success, it is because a reader finds something meaningful to them personally in a poem generated by POS-gram.

The examples generated by N-grams are unsurprisingly bland. The context of a word within a line is almost completely lost; two words separated by two or three tokens have almost nothing to do with each other. In contrast, the POS-gram examples are more likely to be syntactically correct, and more likely to resemble coherent thoughts. 3-grams are more likely than 2-grams to create syntactic, coherent sentences, but are much more likely to repeat themselves or quote the source document verbatim than either 2-grams or POS-grams.

Generally speaking, the more restrictions imposed, the smaller the search space and the more likely to run into problems like repetition. Some repetition is not always bad, though; repetition is often used in human-written poetry. My favorite generated examples are "Sentences and Wordnet, No Rhymes", partly because the occasional repetition creates a sense of theme that runs through the piece.

While rhyming lines are generally more pleasing to read, the extra restrictions imposed restrict the search space. For example, the "Wordnet Relatedness" examples end up using the same couple of rhyming words over and over, making the resulting set of couplets more boring when presented together as a poem. This problem might be mitigated by using a larger source document, which would possibly give a larger set of rhyming words to choose from.

<u>2-Grams</u>
rather glad there was nothing else to you :
she found a little nervous about new ;
but she could not possibly reach into :
the little door into that lovely you .

the little door into that lovely you .
way ? ' , but she could not possibly reach too :
how many miles i 've fallen by this to ?
was nothing so very much out of too ,
as she could not possibly reach into :
neck of the country is , you know , said to ;
she ate a little nervous about new ;
she set to work , and hot buttered toast you)
fell on a little nervous about new ;
no , i wonder what was going to you :
latitude or longitude i 've got do ?
i would n't say anything about into ,

3-Grams
she felt a little nervous about new ;
how many miles i 've fallen by this to ?
larger , i can creep under the into ;
how many miles i 've fallen by this to ?
miss me very much to-night , i should me !
how many miles i 've fallen by this she ?
perhaps i shall see it written up near .
latitude or longitude i 've got hear ?
seemed to be no use in crying like she !
miss me very much to-night , i should me !
miss me very much to-night , i should you !
she might find another key on into ,
miss me very much to-night , i should you !
she felt a little nervous about new ;
she found a little bottle on into ,
the little door into that lovely you .

Blank Verse
for it might belong to one of the house !
if my head would go through , thought poor alice ,)
little bottle on it in large letters .
when the rabbit say to itself , oh dear !
with curiosity , she said aloud .
longitude either , but they were all locked ;
so alice soon began talking again .
shall think nothing of tumbling down stairs !
little door about fifteen inches high :

could , if i only know how to begin .
a rabbit with pink eyes ran close by her .
it was good practice to say it over)
all the right distance--but then i wonder ?
very little use without my shoulders .
glass box that was lying under the door ;
disagree with you , sooner or later .

Default Options
the rabbit-hole went straight on like a white .
could not even get her head though the right ;
alice started to her feet in a fall :
now the right size for going though the hall ;
long passage , not much larger than a few :
and if it makes me grow smaller , i do -
there 's no use now , dinah , tell me the hall :
by a row of lamps hanging from the fall .
having nothing to do that in a game .
glass box that was lying under the flame .
more happened , she found herself in a game :
that it might belong to one of the same .
through , thought alice without pictures or twice ?
alice was not a very good advice ,
grow smaller , i can creep under the house .
going out altogether , like a mouse !

First Words
the rabbit-hole went straight on like a bit .
first , she ran across the field after it ,
i wonder how many miles i 've got to ?
that dark hall , and finding it very few ,
the wise little alice was not a bit :
it did n't much matter which way she put it .
there 's no use now , dinah , tell me the fall :
it was too small , but it was all dark hall ;
the rabbit-hole went straight on like a bit .
what latitude was , or she fell past it .
there 's no use now , dinah , tell me the door :
would be four thousand miles down , down , down , for .
said , and though this was not going to be :
oh , how i wish you were down here with me !

she ought to have wondered at this , i think -
i could , if i only know how to shrink .

Full Sentences
for it might belong to one of the right !
the antipathies , i can reach the white ;
i wonder how many miles i 've got to ?
and if it makes me grow larger , i do -
the rabbit-hole went straight on like a game .
why , i wonder if i fell off the name .
either the locks were too large , or the same ;
the rabbit-hole went straight on like a game !
the rabbit-hole went straight on like a game .
for it might belong to one of the name !
she said , and of having nothing to do :
the rabbit-hole went straight on like a few .
she could see it quite plainly through the key .
oh , how i wish you were down here with me !
the rabbit-hole went straight on like a fall .
it was too small , but it was all dark hall ;

Iambic Emphasis
about it , and when she got to the flame ;
forgotten that , if i fell off the name .
looked up , but it was all dark overhead ;
for showing off her knowledge , as she said ;
was growing , and that 's very like a book .
jar from one of the cupboards as she took ;
long passage , not much larger than a small :
how i wish i could shut up like a fall .
long passage , not much larger than a small :
think it so very much out of the fall :
i shall see it quite plainly through the well !
jar from one of the cupboards as she fell ;
behind it when she went back to the right :
get out of that dark hall , but it was quite :
it did n't much matter which way she put it .
then she looked up , but you might catch a bit ?

Iambic of Trochaic Emphasis
was coming to , but you might catch a bit ?

alice to herself , which way she put it .
she was coming to , but it was too small ;
behind it when she went back to the hall :
makes me grow smaller , i can reach the hot ;
she jumped up on to her , still it was not :
even if i only know how to make .
even if i only know how to take .
wonder if i only know how to say .
this time , as it did n't much matter which way ?
a watch out of the cupboards as she got ;
i shall see it quite plainly through the hot .
oh , how i wish you were down here with you !
it was good practice to say it into)
and to wonder what was going to do :
it was good practice to say it into)

Sentences and Wordnet, No Rhymes
i think i could shut up like telescopes :
she could see it quite plainly through the earth .
and if it makes me grow larger , i think- -
no , i 'll get into the garden at once ;
and if it makes me grow larger , i think- -
either the locks were too large , or the key ;
i must be shutting up like a candle .
for she could see it written up somewhere .
i advise you to leave off this minute !
for she had plenty of time as she passed ;
and if it makes me grow smaller , i think- -
i wonder how many miles i 've got to ?
i think i could shut up like a candle .
but it was too dark to see anything ;
--but i shall see it written up somewhere .
i think i could shut up like telescopes :

Wordnet Relatedness
it , and very soon finished off the fall .
fell off the top of her head though the hall ;
by a row of lamps hanging from the fall .
was beginning to get out of the hall !
could not even get her head though the fall ;
else to do , so she went back to the hall :

and if it makes me grow smaller , i do -
a bit hurt , and finding it very few ,
going out altogether , like a fall .
close behind it when she got to the hall ;
wonder if i only know how to find .
she took down a jar from one of the wind !
she took down a jar from one of the fall !
fell off the top of her head though the hall ;
doors of the doors of the shelves as she fell ;
to the door , so she went back to the well :

Pseudocode for the POS-gram algorithm

I use Pythonic pseudocode to illustrate my implementation of POS-gram and its variants. Colons at the end of a line indicate the beginning of a new code block or level of scope in the next line. Indentation indicates the level of scope. "{}" indicates an empty associative array. Comma-separated items within parenthesis indicate in immutable list; square brackets indicate a mutable list. Parenthesis appended to a token indicate a function call; square brackets indicate list element access or associative map lookup. "#" indicates a comment.

Naturally, there are inefficiencies in this code that are left unoptimized for the sake of code clarity. In the true implementation, I cache data in various ways to avoid unnecessary recomputation and speed execution.

```
# Function generate_couplet
#
# @input source
# The source document to imitate
#
# @returns couplet
# Two lines of poetry

1  function generate_couplet(input):
2
3    # Tag the input
4    tagged_input = pos_tag(input)
5
6    # Map pos-1, word-1 onto current pos
7    pos_map_1_1 = {}
8
9    # Map pos-1, word-1, current pos onto current word
10   word_map_2_1 = {}
11
12   # Map pos-2, word-2, pos-1, word -1 onto current pos
```

```
pos_map_2_2 = {}

# Map pos-2, word-2, pos-1, word -1, current pos to current word
word_map_3_2 = {}

# Build maps
for tagged_sentence in tagged_input:
  for (word, pos) in tagged_sentence:
    pos_map_1_1[pos-1, word-1].append(pos)
    word_map_2_1[pos-1, word-1, pos].append(word)
    pos_map_2_2[pos-2, word-2, pos-1, word-1].append(pos)
    word_map_3_2[pos-2, word-2, pos-1, word-1, pos].append(word)

while True:
  line_1, tags_1 = generate_line(pos_map_1_1, word_map_2_1,
                                 pos_map_2_2, word_map_3_2)
  line_2, tags_2 = generate_line(pos_map_1_1, word_map_2_1,
                                 pos_map_2_2, word_map_3_2)

  # Find rhyming words that work at the ends of the lines
  rhymes = []
  for word_1 in word_map_2_1[tags_1[-2], line_1[-2], tags_1[-1]]:
    for word_2 in word_map_2_1[tags_2[-2], line_2[-2],
                               tags_2[-1]]:
      if words_rhyme(word_1, word_2):
        rhymes.append((word_1, word_2))
  if len(rhymes) == 0:
    continue
  line_1[-1], line_2[-1] = choice(rhymes)

  # Enforce line length, using whole sentences
  if syllables(line_1) != 10 or syllables(line_2) != 10:
    continue

  # Enforce iambic pentameter
  iambic = True
  for line in (line_1, line_2):
    for i in range(1, 10): # From 1 to 9
      if i % 2 == 0:
        if emphasis(line, i) > emphasis(line, i-1):
          iambic = False
      else:
        if emphasis(line, i) < emphasis(line, i-1):
          iambic = False
  if not iambic:
    continue

  # Enforce line relatednes
  score = 0
```

```
    for word in line_1:
      for word in line_2:
        if words_are_related(word_1, word_2):
          score += 1
    if score < 2:
      continue

    return (line_1, line_2)

# Function generate_line
#
# @inputs pos_map_1_1, word_map_2_1, pos_map_2_2, word_map_3_2
# Tables with information about word and POS distributions
#
# @returns words, parts_of_speech
# A single line of poetry and its corresponding parts of speech

 function generate_line(pos_map_1_1, word_map_2_1,
                        pos_map_2_2, word_map_3_2):

   # Line length in tokens, shorter when you're not going to trim it later
   line_length = whole_sentences ? 14 : 20
   words = ['START']
   poss = ['START'] # Parts of speech

   for i in range(1, line_length):

     if poss[-1] == 'END':
       return words, poss

     # Add a POS token
     if i >= 2
         and (poss[i-2], words[i-2], poss[i-1], words[i-1]) in pos_map_22:
       poss.append(choice(
           pos_map_22[poss[i-2], words[i-2], poss[i-1], words[i-1]))
     else if (poss[i-1], words[i-1]) in pos_map_1_1:
       poss.append(choice(pos_map_1_1[poss[i-1], words[i-1]]))
     else: # No more parts of speech found for this sequence
       return words, poss

     # Add a word
     if i >= 2
         and (poss[i-2], words[i-2], poss[i-1], words[i-1], poss[i])
         in word_map_3_2:
       words.append(choice(
           word_map_3_2[poss[i-2], words[i-2], poss[i-1],
                        words[i-1], poss[i]]))
     else if (poss[i-1], words[i-1], pos[i]) in word_map_2_1:
       words.append(choice(
```

```
33            word_map_2_1[poss[i-1], words[i-1], pos[i]]))
34      else: # No more words for for this sequence
35        return words, poss
36
37    return words, poss

# Function words_are_related
#
# @inputs word_1, word_2
# Words to be compared
#
# @returns
# True if the words are related,
# otherwise False

1   function words_are_related(word_1, word_2):
2     if word_1 == word_2:
3       return False
4     for (word, other) in permutations(word_1, word_2):
5       synsets = synsets(word)
6       for other_synset in synsets(other):
7         if other_synset in synsets:
8           return True
9         for hypernym in hypernyms(synsets):
10          if other_synset in hypernyms:
11            return True
12        for hyponym in hyponyms(synsets):
13          if other_synset in hyponyms:
14            return True
15    return False
```

Bibliography

Yael Netzer, David Gabay, Yoav Goldberg, and Michael Elhada, *Gaiku : Generating Haiku with Word Associations Norms*. CALC-2009 (NAACL Workshop on Computational Approaches to Linguistic Creativity).