Pyambic Pentameter

Generating rhyming and metered poems with Markov chains and NLTK

Kathryn Lingel

Pyambic Pentameter

Generating rhyming and metered poems with Markov chains and NLTK

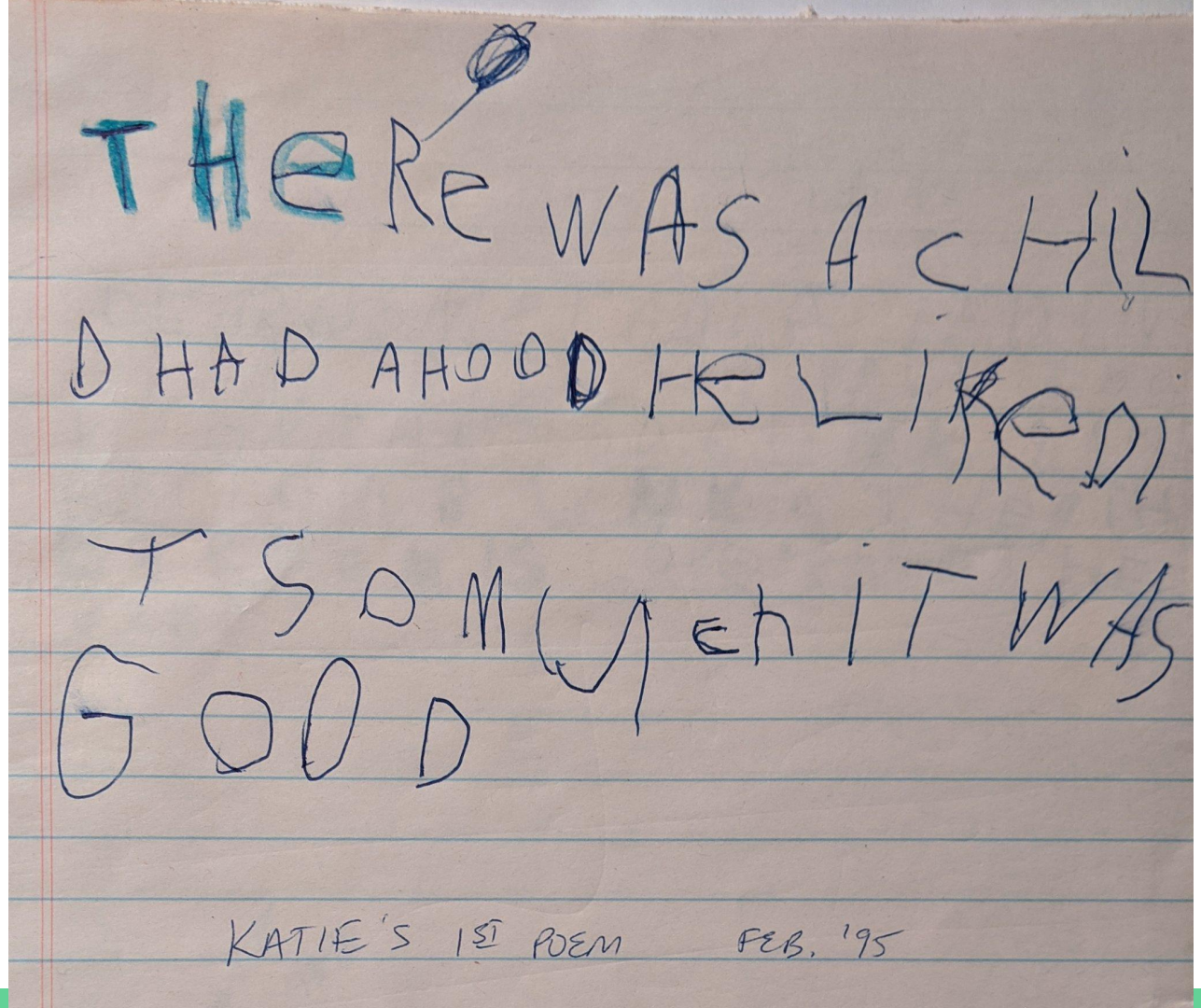
Kathryn Lingel

Poetry 101

for computers

Poetry 101

for humans


THERE WAS A CHILD
D HAD A HOOD HE LIKED
T SOMETHING IT WAS
GOOD
KATIE'S 1ST POEM FEB. '95

Limericking
@Limericking

We all love the goose in the game
Who's recently waddled to fame,
A nihilist force
Devoid of remorse,
A bird without pity or shame.

11:48 AM · Sep 29, 2019 · TweetDeck

honk

18.

SHall I compare thee to a Summers day?
Thou art more louely and more temperate:
Rough windes do ſhake the darling buds of Maie,
And Sommers leaſe hath all too ſhort a date:
Sometime too hot the eye of heauen ſhines,
And often is his gold complexion dimm'd,
And euery faire from faire ſome-time declines,
By chance, or natures changing courſe vntrim'd:
But thy eternall Sommer ſhall not fade,
Nor looſe poſſeſſion of that faire thou ow'ſt,
Nor ſhall death brag thou wandr'ſt in his ſhade,
When in eternall lines to time thou grow'ſt,
So long as men can breath or eyes can ſee,
So long liues this, and this giues life to thee,

Rhyme

Rhyme
&
Meter

Generating Text

Generating Text

Markov-style 

I am the egg man
They are the egg men
I am the walrus
Goo goo g’joob

-John Lennon

I am the egg man

They are the egg men

I am the walrus

Goo goo g’joob

I am the egg man

They are the egg men

I am the walrus

Goo goo g’joob

I am

I am the egg man
They are the egg men
I am the walrus
Goo goo g’joob

I am the

I am the egg man
They are the egg men
I am the walrus
Goo goo g’joob

I am the egg

I am the egg man
They are the egg men
I am the walrus
Goo goo g’joob

I am the egg men

```
def build_model(source_text):
```

```
def build_model(source_text):
    list_of_words = source_text.split()
```

```
def build_model(source_text):
    list_of_words = source_text.split()
    model = {}
```

```
def build_model(source_text):
    list_of_words = source_text.split()
    model = {}

    for i, word in enumerate(list_of_words[:-1]):
```

```
def build_model(source_text):
    list_of_words = source_text.split()
    model = {}

    for i, word in enumerate(list_of_words[:-1]):
        if not word in model:
            model[word] = []
```

```
def build_model(source_text):
    list_of_words = source_text.split()
    model = {}

    for i, word in enumerate(list_of_words[:-1]):
        if not word in model:
            model[word] = []
        next_word = list_of_words[i+1]
```

```
def build_model(source_text):
    list_of_words = source_text.split()
    model = {}

    for i, word in enumerate(list_of_words[:-1]):
        if not word in model:
            model[word] = []
        next_word = list_of_words[i+1]
        model[word].append(next_word)
```

```
def build_model(source_text):
    list_of_words = source_text.split()
    model = {}

    for i, word in enumerate(list_of_words[:-1]):
        if not word in model:
            model[word] = []
        next_word = list_of_words[i+1]
        model[word].append(next_word)

    return model
```

```
> build_model("i am the egg man they are the egg men i am the walrus goo goo g'joob")
```

```
> build_model("i am the egg man they are the egg men i am the walrus goo goo g'joob")

{'i': ['am', 'am'],
 'am': ['the', 'the'],
 'the': ['egg', 'egg', 'walrus'],
 'egg': ['man', 'men'],
 'man': ['they'],
 'they': ['are'],
 'are': ['the'],
 'men': ['i'],
 'walrus': ['goo'],
 'goo': ['goo', "g'joob"]}
```

```
import random

def markov_generate(source_text, num_words=20):
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
    for i in range(num_words):
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
    for i in range(num_words):
        last_word = output[-1]
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
    for i in range(num_words):
        last_word = output[-1]
        next_word = random.choice(model[last_word])
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
    for i in range(num_words):
        last_word = output[-1]
        next_word = random.choice(model[last_word])
        output.append(next_word)
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
    for i in range(num_words):
        last_word = output[-1]
        next_word = random.choice(model[last_word])
        output.append(next_word)
        if next_word not in model:
            break
```

```
import random

def markov_generate(source_text, num_words=20):
    model = build_model(source_text)
    seed = random.choice(list(model.keys()))
    output = [seed]
    for i in range(num_words):
        last_word = output[-1]
        next_word = random.choice(model[last_word])
        output.append(next_word)
        if next_word not in model:
            break

    return ' '.join(output)
```

```
> markov_generate("i am the egg man they are the
egg men i am the walrus goo goo g'joob")
```

```
> markov_generate("i am the egg man they are the egg men i am the walrus goo goo g'joob")

"the egg men i am the egg man they are the walrus goo goo goo g'joob"
```

```
> markov_generate("i am the egg man they are the
egg men i am the walrus goo goo g'joob")

"men i am the walrus goo g'joob"
```

```
> markov_generate("i am the egg man they are the
egg men i am the walrus goo goo g'joob")

"goo goo goo goo goo goo goo g'joob"
```

NLTK

Natural Language ToolKit

Carnegie Mellon Pronouncing Dictionary

`nltk.corpus.cmudict`

```
> from nltk.corpus import cmudict
> dictionary = cmudict.dict()
> dictionary['python']
```

```
> from nltk.corpus import cmudict
> dictionary = cmudict.dict()
> dictionary['python']

[['P', 'AY1', 'TH', 'AA0', 'N']]
```

```
> from nltk.corpus import cmudict
> dictionary = cmudict.dict()
> dictionary['separate']
```

```
> from nltk.corpus import cmudict
> dictionary = cmudict.dict()
> dictionary['separate']

[['S', 'EH1', 'P', 'ER0', 'EY2', 'T'],
 ['S', 'EH1', 'P', 'ER0', 'IH0', 'T'],
 ['S', 'EH1', 'P', 'R', 'AH0', 'T']]
```

Poetry 101

for computers

Rhyme

Rhyme

...just go backwards!

```
{('OW1',): ['grow', 'row', 'poe', 'ho'],
  ('AH1', 'N'): ['run', 'gun', 'sun'],
  ('AH1', 'M'): ['from', 'come'],
  ('AY1',): ['i', 'eye', 'sky', 'fly', 'sty'],
  ('UW1',): ['you', 'to', 'goo', 'do'],
  ('EH1', 'T'): ['get', 'let'],
  ('IY1',): ['he', 'hee', 'see', 'me', 'we'],
  ('AO1', 'R'): ['for', 'your'],
  ('AE1', 'N'): ['man', 'an', 'van', 'tan'],
  ('IH1', 'T', 'IY0'): ['city', 'pretty']})
```

```
{('OW1',): ['grow', 'row', 'poe', 'ho'],
  ('AH1', 'N'): ['run', 'gun', 'sun'],
  ('AH1', 'M'): ['from', 'come'],
  ('AY1',): ['i', 'eye', 'sky', 'fly', 'sty'],
  ('UW1',): ['you', 'to', 'goo', 'do'],
  ('EH1', 'T'): ['get', 'let'],
  ('IY1',): ['he', 'hee', 'see', 'me', 'we'],
  ('AO1', 'R'): ['for', 'your'],
  ('AE1', 'N'): ['man', 'an', 'van', 'tan'],
  ('IH1', 'T', 'IY0'): ['city', 'pretty']})
```

```
{('OW1',): ['grow', 'row', 'poe', 'ho'],
  ('AH1', 'N'): ['run', 'gun', 'sun'],
  ('AH1', 'M'): ['from', 'come'],
  ('AY1',): ['i', 'eye', 'sky', 'fly', 'sty'],
  ('UW1',): ['you', 'to', 'goo', 'do'],
  ('EH1', 'T'): ['get', 'let'],
  ('IY1',): ['he', 'hee', 'see', 'me', 'we'],
  ('AO1', 'R'): ['for', 'your'],
  ('AE1', 'N'): ['man', 'an', 'van', 'tan'],
  ('IH1', 'T', 'IY0'): ['city', 'pretty']})
```

```
{('OW1',): ['grow', 'row', 'poe', 'ho'],
  ('AH1', 'N'): ['run', 'gun', 'sun'],
  ('AH1', 'M'): ['from', 'come'],
  ('AY1',): ['i', 'eye', 'sky', 'fly', 'sty'],
  ('UW1',): ['you', 'to', 'goo', 'do'],
  ('EH1', 'T'): ['get', 'let'],
  ('IY1',): ['he', 'hee', 'see', 'me', 'we'],
  ('AO1', 'R'): ['for', 'your'],
  ('AE1', 'N'): ['man', 'an', 'van', 'tan'],
  ('IH1', 'T', 'IY0'): ['city', 'pretty']})
```

'corn flake waiting for the **sky**'
'egg man you've been a **sty**'

Meter

Meter

guess-and-check; backtrack if necessary

desired: 0101010101

```
desired: 0101010101
found:   ----------
```

```
desired: 0101010101
found:   ----------
```

little

```
> dictionary['little']
[['L', 'IH1', 'T', 'AH0', 'L']]
```

```
desired: 0101010101
found:   10--------

1  0
little
```

```
desired: 0101010101
found:   10--------
         XX
```

1 0

little

```
desired: 0101010101
found:   0---------

0
i'm
```

desired: 0101010101
found: 010-------

0   1 0
i’m crying

desired: 0101010101
found: 01010-----

0   1 0   1 0
i’m crying yellow

```
desired: 0101010101
found:   0101010---

0   1   0   1   0   1   0
i'm crying yellow matter
```

desired: 0101010101
found: 010101010-

0 1 0 1 0 1 0 1 0
i’m crying yellow matter custard

```
desired: 0101010101
found:   01010101010
                    X
0   1   0   1   0   1   0   1   0   1   0
i’m crying yellow matter custard dripping
```

desired: 0101010101
found: 010101010-

0 1 0 1 0 1 0 1 0
i’m crying yellow matter custard

```
desired: 0101010101
found:   0101010---

0   1   0   1   0   1   0
i'm crying yellow matter
```

desired: 0101010101
found: 01010-----

0   1 0   1 0
i’m crying yellow

desired: 0101010101
found: 010-------

0   1 0
i’m crying

```
desired: 0101010101
found:   01010-----

0   1  0   1   0
i’m crying sitting
```

desired: 0101010101
found: 010101----

0 1 0 1 0 1
i’m crying sitting in

desired: 0101010101
found: 0101010---

0 1 0 1 0 1 0
i’m crying sitting in the

```
desired: 0101010101
found:   01010101--

0   1  0   1   0   1  0   1
i’m crying sitting in the sky
```

desired: 0101010101
found: 010101011-

0 1 0 1 0 1 0 1 1
i’m crying sitting in the sky see

desired: 0101010101
found: 010101011-
                  X
0 1 0 1 0 1 0 1 1
i’m crying sitting in the sky see

```
desired: 0101010101
found:   01010101--

0   1  0   1   0   1  0   1
i’m crying sitting in the sky
```

desired: 0101010101
found: 0101010---

0 1 0 1 0 1 0
i’m crying sitting in the

```
desired: 0101010101
found:   010101010-

0   1  0   1   0   1  0   1  0
i’m crying sitting in the walrus
```

desired: 0101010101
found: 0101010101

0 1 0 1 0 1 0 1 0 1
i’m crying sitting in the walrus goo

desired: 0101010101
found: 0101010101 ✓

0 1 0 1 0 1 0 1 0 1
i’m crying sitting in the walrus goo

All Together Now

(that’s a Beatles joke)

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
    seeds = random.choices(rhymes[chosen_rhyme], k)
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
    seeds = random.choices(rhymes[chosen_rhyme], k)

    output = []
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
    seeds = random.choices(rhymes[chosen_rhyme], k)

    output = []
    for seed in seeds:
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
    seeds = random.choices(rhymes[chosen_rhyme], k)

    output = []
    for seed in seeds:
        line = generate_backwards_with_meter(seed, model, meter)
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
    seeds = random.choices(rhymes[chosen_rhyme], k)

    output = []
    for seed in seeds:
        line = generate_backwards_with_meter(seed, model, meter)
        output.append(line)
```

```
import random

def generate_simple_poem(source_text, meter='01' * 5, k=2):
    model = build_backwards_model(source_text)
    rhymes = find_rhymes(source_text, k)
    chosen_rhyme = random.choice(list(rhymes.keys()))
    seeds = random.choices(rhymes[chosen_rhyme], k)

    output = []
    for seed in seeds:
        line = generate_backwards_with_meter(seed, model, meter)
        output.append(line)

    return '\n'.join(output)
```

Live Demo!

(don’t worry, I have a backup plan)

Thank you!

I am @hartknyx

Neverending poetry at https://poems.katlings.net/

Source and slides at https://github.com/katlings/pyambic-pentameter

Acknowledgements

Inspired by @pentametron 🤖

Speaker mentors: Hanna Landrus, Al Sweigart

First-time speaker resources from PyCascades and Pyladies

PuPPy

Recommended Reading

- A discussion of poetry in different languages
  http://stories.schwa-fire.com/language_of_poetry
- The intro-to-CS assignment that taught me about Markov generation
  https://www.cs.hmc.edu/twiki/bin/view/CSforAll/MarkovText1
- Pycon Israel 2016: “Poetry in Python: Using Markov Chains to Generate Texts” by Omer Nevo https://www.youtube.com/watch?v=yyNTjDkQQEk
- The Mechanical Muse, a New Yorker article about machine-generated sonnets
  https://www.newyorker.com/culture/annals-of-inquiry/the-mechanical-muse

ooo baby twist and nothing with his socks
that all together love of what a whim
was wrong is working for this letter box
decide to sit and nearly made a trim

him oh it comes before to have to find
with and does her because the albert hall
the evening wanna dance to write this kind
does don't be sending me this world was paul

or night and windy night just sees the wild
i'm still believe and start that taste of lords
has fun tonight and our little child
this bird is gonna change and plays the chords

with me and kept her words to realize
it took his world that i'll apologize