

Replace yourself with a very small shell script

Stefanie Schirmer
@linse

this slide is just for the clock

Replace yourself with a very small shell script

Stefanie Schirmer
@linse

Hi, I am Steffi and I still work at [etsy.com](https://www.etsy.com). :D

To make my life easier, I'd like to replace myself with a very small shell script.

Replace yourself with a very small shell script

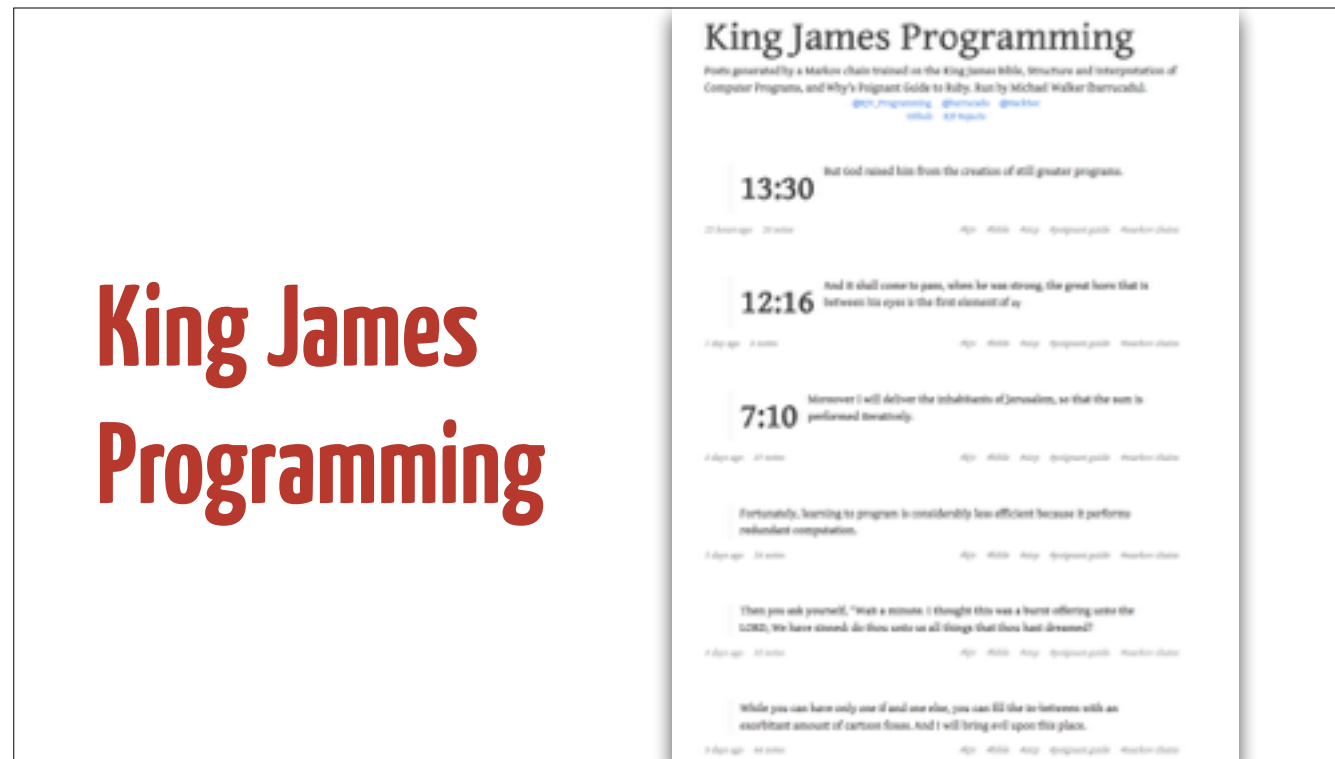
~~Stefanie Schirmer~~
~~@linse_ebooks~~

This is linse ebooks, and she's taking over from now on.



Recently, there has been an outbreak of markov bots.

King James Programming



There is for example the king james programming site, where somebody mixed the king james bible, structure and interpretation of computer programs, and some ruby guide.

the mix brings up something like:

while you can have only one ifa nd else, you can fill the in-between with an exorbitant amount of cartoon foxes. and I will bring evil upon this place.

The Doom that came to Puppet

The Doom that Came to Puppet

Posts generated by a Markov chain trained on the Puppet documentation and the assorted works of H. P. Lovecraft. Created by @bransen using bernacchi/markov. Inspired by King James Programming.

"Replace 'namegoeshere' with the function name, and even if the most drastic directions were not carried out, he must be placed where he could inflict no harm upon Charles Ward."

11 months ago

"Things-presences or voices of some sort-could be drawn down from unknown places as well as to standard out."

11 months ago

"Some of the upper levels were wholly vacant, but most of the space was filled with small odd-looking leaden jars of two general types; one tall and without handles like a Grecian lekythos or oil-jug, and the other with a single resource type"

11 months ago

There is a mix of the works of HP Lovecraft and the puppet manual:

Things-presences or voices of some sort could be drawn down from unknown places as well as to standard out.

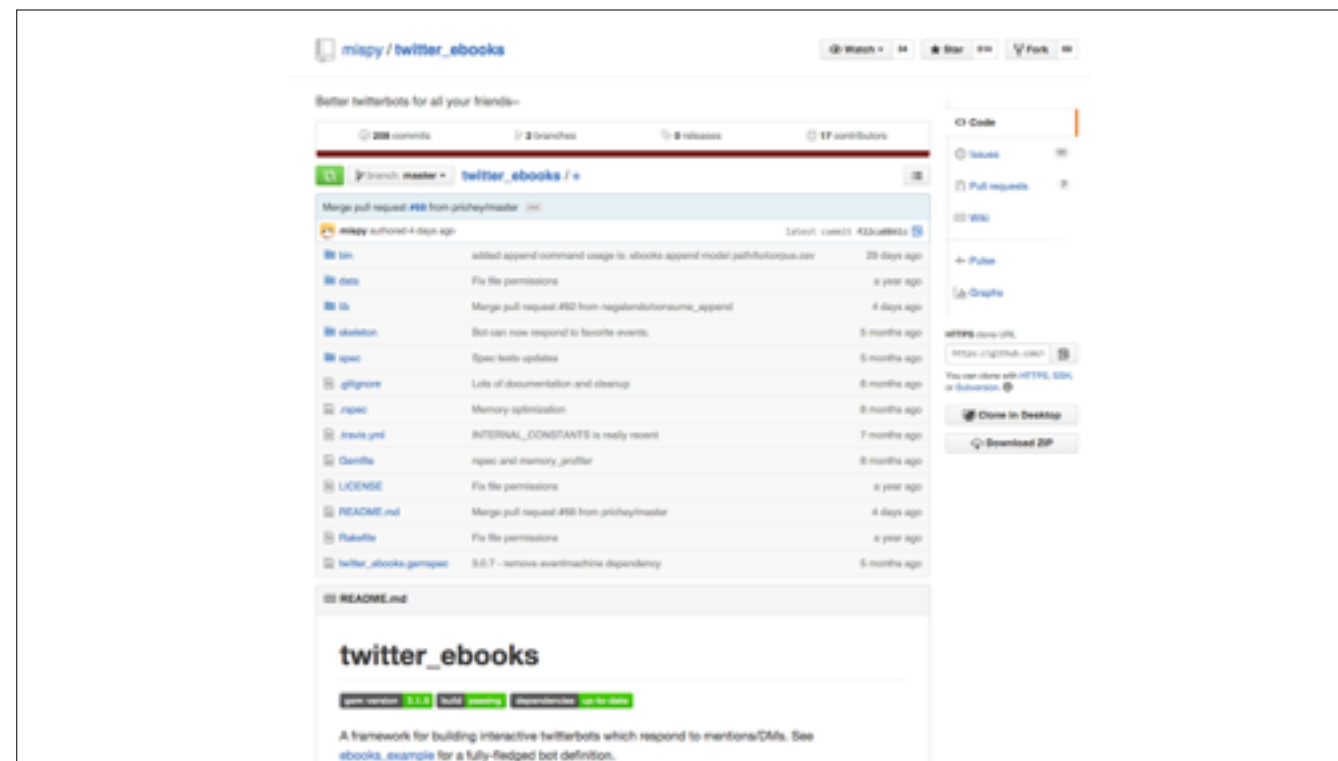
Erowid Recruiter



And then there is the Erowid Recruiter, which is a mix of drug use reports and recruiter email spam. PHP, Python or Perl, years of experience with psychedelic hallucinations.

The easy way

For my bot on twitter, linse_ebooks, I used the easy way. A premade library from user “abadidea” on twitter.



On github it is mispy - twitter-underscore-ebooks.

A ruby script can be used to download all your tweets, consume them and build a model, and produce new tweets.

You just have to enter your and the bot's account details and run the script.

How does it work?

But of course I wanna know how this works.

In my life before Etsy I tried to predict the foldings of molecules with a statistical model. Maybe we can build such a model for writings by humans?

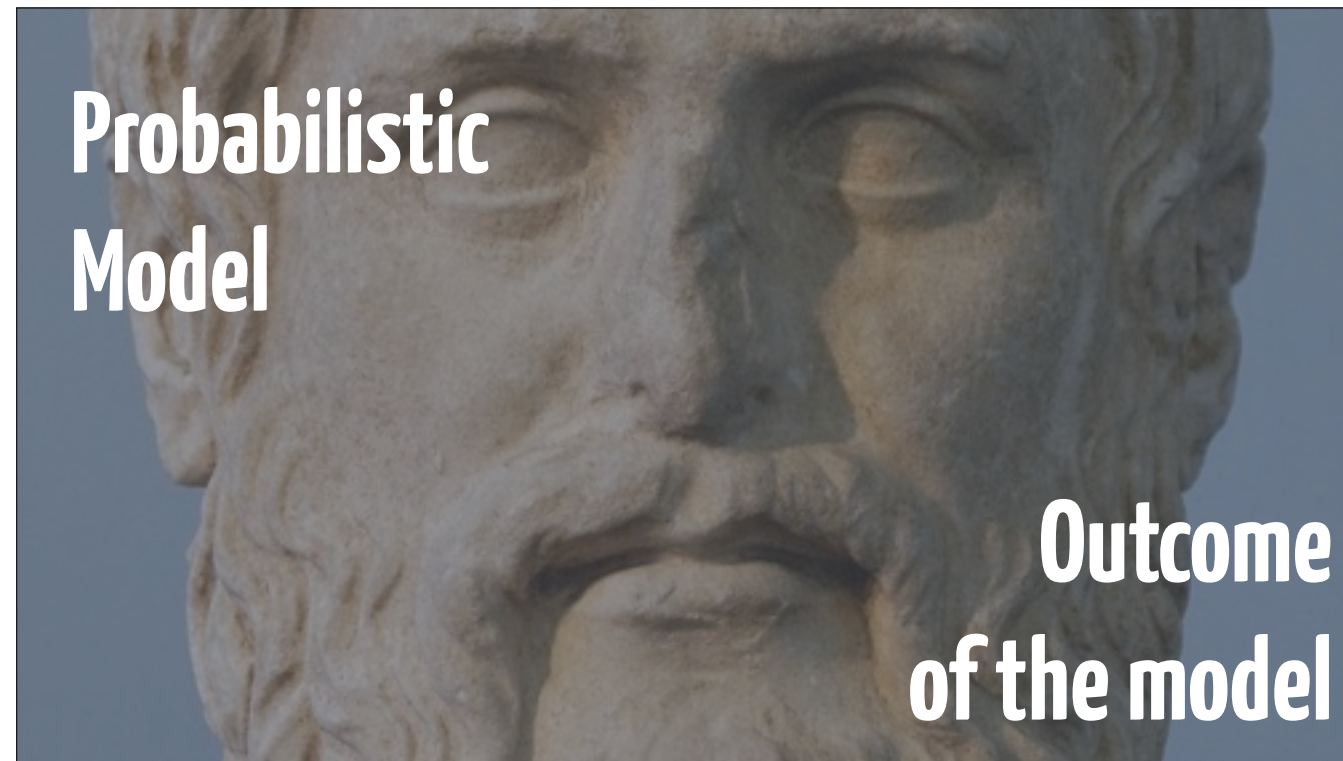


**Plato:
Ideas are hidden**

**Real things are
all a bit different**

Let's time travel a bit.

Plato is well known for his theory of the world of ideas. Think of the idea of a perfect circle, it's totally possible for us to imagine it, but it's impossible for me to draw one. Each real one will be a bit different.



We could say the perfect circle is the idea, the model.

The outcome is what we see in the real world.

Can we do this for text? What's the model for text? Or for a sentence?

Model for text: **n-gram** probability

One of the simplest, but also very effective features to build a text model is the probability of words appearing in sequence. An n-gram is a subsequence of length n. For text this means two words in a row if we choose n to be two. This is also called a bi-gram.

What is the probability of the word apple followed by the word sauce?

What is the probability of the word apple followed by the word strudel?

What is the probability of word A followed by word B?

```
#!/usr/bin/python
```

printBigrams.py

```
import sys
```

```
fname = sys.argv[1]
```

```
previous = ''
```

```
with open(fname) as f:
```

```
    for line in f:
```

```
        for word in line.split():
```

```
            if previous != '':
```

```
                print previous, word
```

```
            previous = word
```

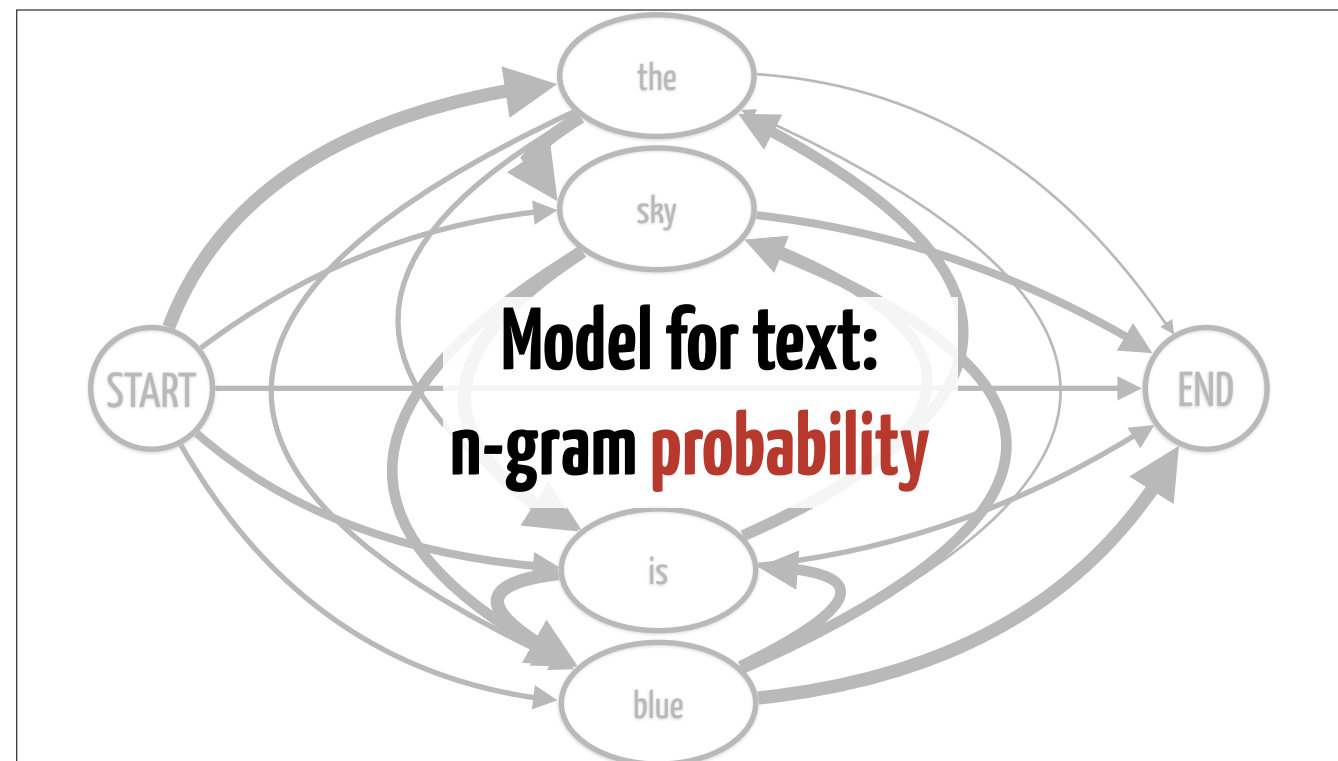
Let's print out all bi-grams from the text.

We keep track of the previous word, and then it's just a matter of looping through word by word, printing out the previous and the current word.

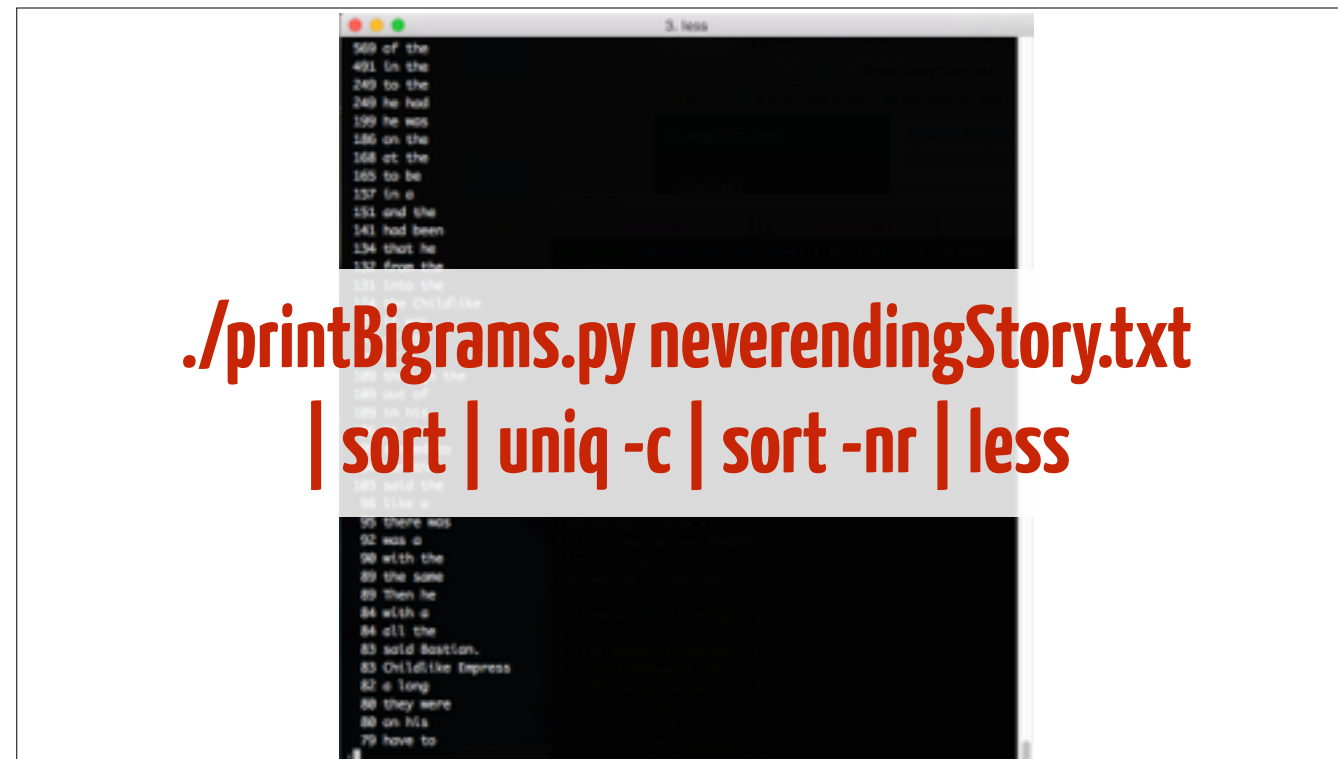
Model for text: n-gram **probability**

We should sort them, and count them, to compute transition probabilities.

This is the learning step of our algorithm.



With the transition probabilities, we can imagine our text corpus like this weighted graph. Some paths through the graph are more or less likely to be found in real text. So let's count the bi-grams.

A terminal window titled '3. less' displays the output of a script. The output consists of two columns of text, each preceded by a line number. The first column contains bigrams like 'of the', 'in the', 'to the', 'he had', 'he was', 'on the', 'at the', 'to be', 'in a', 'and the', 'had been', 'that he', 'from the', 'into the', 'like the', 'like a', 'there was', 'was a', 'with the', 'the same', 'Then he', 'with a', 'all the', 'said Boston.', 'Childlike Express', 'a long', 'they were', 'on his', and 'have to'. The second column contains bigrams like 'the of', 'the in', 'the to', 'the he', 'the he', 'the on', 'the at', 'the to', 'the in', 'the and', 'the had', 'the that', 'the from', 'the into', 'the like', 'the like', 'the there', 'the was', 'the with', 'the the', 'the Then', 'the with', 'the all', 'the said', 'the Childlike', 'the Express', 'the a', 'the they', 'the on', and 'the have'. The terminal window has a dark background and a light-colored border.

```
569 of the
491 in the
249 to the
249 he had
199 he was
186 on the
168 at the
165 to be
137 in a
131 and the
141 had been
134 that he
132 from the
131 into the
131 like the
131 like a
99 there was
92 was a
90 with the
89 the same
89 Then he
84 with a
84 all the
83 said Boston.
83 Childlike Express
82 a long
80 they were
80 on his
79 have to
```

**`./printBigrams.py neverendingStory.txt
| sort | uniq -c | sort -nr | less`**

Unix tools to the rescue! I can get a good first glance of the distribution by piping the bigrams into `sort | uniq -c | sort -nr | less`

record bi-grams -> model
check for sentences

```
def allBigrams(bigrams, fname):  
    previous = ''  
    with open(fname) as f:  
        for line in f:  
            for word in line.split():  
                if previous != '':  
                    if previous.endswith(('.', '?', '!')):  
                        bigrams = addBigram(bigrams, previous, 'END')  
                        bigrams = addBigram(bigrams, 'START', word)  
                    else:  
                        bigrams = addBigram(bigrams, previous, word)  
                previous = word  
    return bigrams
```

To put all bigrams into a model data structure, instead of printing them, we use a dictionary called bigrams. The looping is the same as before.

Also, we take care of sentence beginnings and endings. We add a START and END state to indicate those. Adding START and END state makes a huge difference in how real the output sounds - because we mimic a sentence structure!

record one bi-gram

```
def addBigram(model, first, second):  
    if not first in model:  
        model[first] = {}  
    if not second in model[first]:  
        model[first][second] = 1  
    else:  
        model[first][second] = model[first][second]+1  
    return model
```

How do we add a new bi-gram to our model?

The model is a two dimensional lookup structure, here a python dictionary.

It is indexed by the first and second word of the bi-gram. And it gives us the number of occurrences of this particular two words following each other in the text. For a new first word, we set up a new dict for that word.

For a new bigram, we set up a new dict entry with count one.

For a known bigram, we increment it's count in the model.



Ok, our model is ready now!

How do we make it talk?

To generate text, we have to traverse the states of our model, according to the probabilities of the transitions.

Starting from a START state, I could travel into a "Hello" state, into a "there" state and an exclamation mark state and arrive at an END state, for example.

generate a sentence

```
def main():  
    # 1. learn model  
    model = {}  
    # skip program name  
    for arg in sys.argv[1:]:  
        model = allBigrams(model, arg)  
    # 2. generate  
    state = 'START'  
    while state != 'END':  
        state = step(model, state)
```

How do we write this in code?

Here you see we prepared the model. Here we can add a second input file.

Then, for the generation part, we start from the Start state.

And we make steps until we reach the end state.

The cool thing is that with our bi-gram model we don't need to know anything besides the current state, in order to generate the next state. We have no knowledge of the past. This property is called memorylessness, or also called "the Markov property". This keeps the model simple and elegant.

find next word

```
def step(model, state):  
    nextStates = model[state].items()  
    nextState = weighted_choice(nextStates)  
    if not nextState=='END':  
        print nextState,  
    return nextState
```

In each step we pick a next state or word according to the distribution of the current word's successors that we counted in the text.

Let's make the weighted choice.

find next word

```
def weighted_choice(choices):  
    total = sum(w for word, w in choices)  
    r = random.uniform(0, total)  
    upto = 0  
    for word, w in choices:  
        if upto + w > r:  
            return word  
        upto += w  
    assert False, "Shouldn't get here"
```

We sample (or we could say choose or draw) the next word according to the distribution, depending on the number of words that can be reached from a word and how often they are seen.

By drawing randomly from the sum of all seen next words (summing up all counts), we can make a choice weighted by those counts.



Let's try it out!

So let's run our script and produce a sentence!

Starting from the Start state, we make steps until the sentence is complete and we have reached the end state.

Thank you!

I'm Stefanie Schirmer
sschirme@gmail.com
[@linse](#) on twitter