Node Markov

Download exercise

In this exercise, you'll use an algorithm for generating realistic machine-made text from an original source text.

For example, if we feed in the source text of "Alice's Adventures in Wonderland", we might get output like this:

> 'You are old,' said the Dormouse, who was talking. Alice could only see her. She is such a new pair of white kid gloves and the blades of grass, but she remembered the number of bathing machines in the kitchen that did not like the wind, and was just beginning to grow up any more if you'd like it put the Dormouse again, so she went nearer to make out that it was certainly English. 'I don't quite understand you,' she said, 'for her hair goes in such confusion that she was looking down with it.

(That text isn't directly in the source book, but it's built via an algorithm to be similar to the text in the book).

Markov Machines

A Markov Machine emits output of a "Markov Chain."

A Markov Chain is a chain of possible outcomes, given a particular "state".

For example, consider the phrase "the cat in the hat is in the hat". We could build a table of each word in this phrase, along with the word that comes after that phrase:

the	cat, hat, hat
cat	in
in	the, the
hat	is, ***null***
is	in

To emit realistic-but-random text, we could pick a starting word randomly (say, "in"). Then we would:

1. find all words that can come after that word
2. pick one of those next-words randomly
3. if we picked ***null*** , we've reached the end of the chain, so stop
4. otherwise, restart at step 1

For example, from that simple phrase, we could find:

- "in the cat in the hat"
- "in the hat is in the hat"
- "in the cat in the cat in the cat in the hat"

Step 0: Setup

- Make a project, a Git repo, and a ***package.json***
- Add ***node_modules*** to your ***.gitignore***

Step 1: Implement the Markov Machine

We've given you the start of an object-oriented Markov machine:

markov.js

```
/** Textual markov chain generator */ class MarkovMachine { /** build markov machine; read in text.*/ constructor(text) { let words = text.split(/[ \r\n]+/); this.words = words.filter(c => c !== ""); this.makeChains(); } /** set markov chains: * * for text of "the cat in the hat", chains will be * {"the": ["cat", "hat"], "cat": ["in"], "in": ["the"], "hat": [null]} */ makeChains() { // TODO } /** return random text from chains */ makeText(numWords = 100) { // TODO } }
```

The ***constructor*** function contains some code to get you started—given some input text, it splits it on spaces and linebreak characters to make a list of words. It then calls the (unimplemented) function which builds a map of chains of *word → possible-next-words*.

You should be able to instantiate it like this:

```
let mm = new MarkovMachine("the cat in the hat");
```

Then, whenever you want to get generated text from it:

```
mm.makeText(); mm.makeText(numWords=50);
```

Test this in the Node REPL before continuing! We've given you some text files to play with; you can feed these (or parts of these) into your machine to make sure it works.

- Write tests — which is tricky, since this algorithm is inherently random. There are things you can test, though — what are they?

Step 2: Build the *makeText.js* Script

We want a script, ***makeText.js***, that works like this:

```
$node makeText.js file eggs.txt ... generated text from file 'eggs.txt' ...
$node makeText.js url http://www.gutenberg.org/files/11/11-0.txt ... generate
d text from that URL ...
```

Make sure it handles errors (can't read file or can't read URL) by printing a nice and complete error message and quitting program.

Further Study

There are lots of things you could think of to improve your machine; feel free to pick other things:

Algorithmic Things

- Have the machine only start on a capitalized word (or better still: a word that starts a sentence); this will give you more realistic output text.
- Have the machine stop at a period, while still honoring the maximum number of words passed in.
- Right now, the "state" of your Markov chain is a single word. This generates quite random-sounding text. You can get better text if you deal with *bigrams* — two words at a time. So, you keep track of the two words most recently emitted, and what word could follow that. For the phrase "the cat in the hat is in the hat", this could look like:

the cat	in
cat in	the
in the	hat, hat
the hat	is, ***null***
hat is	in
is in	the

Node Practice Things

With longer text, this can produce quite realistic output text.

- If you give this a URL that returns HTML, you'll get HTML mixed into the output text. Find a library in NPM that can strip out HTML and use this.
- Let the user passing multiple files and/or URLs and make a machine that mixes together that text.

JavaScript Generator Functions

So far, your Markov machines have generated all their text at once. It could be useful, though, to make a machine that generates text on demand, word-by-word, never doing more work than needed.

JavaScript has *generator functions* which can *yield* output on demand. This would allow your machine to be asked to produce output word-by-word.

To do this, you'll need to do some research on generator functions and the ***yield*** keyword.

Solution

View our Solution