>

CodeKata

Because experience is the *only* teacher

- RSS

Search

Navigate…

- PragDave
- Kata
- Archives

Kata14: Tom Swift Under the Milkwood

Trigrams can be used to mutate text into new, surreal, forms. But what heuristics do we apply to get a reasonable result?

As a boy, one of my treats was go to the shops on a Saturday and spend part of my allowance on books; for a nine-year old, I had quite a collection of Tom Swift and Hardy Boys. Wouldn't it be great to be able to create more and more of these classic books, to be able to generate a new Tom Swift adventure on demand?

OK, perhaps not. But that won't stop us trying. I coded up a quick program to generate some swash-buckling scientific adventure on demand. It came up with:

> …it was in the wind that was what he thought was his companion. I think would be a good one and accordingly the ship their situation improved. Slowly so slowly that it beat the band! You'd think no one was a low voice. "Don't take any of the elements and the inventors of the little Frenchman in the enclosed car or cabin completely fitted up in front of the gas in the house and wringing her hands. "I'm sure they'll fall!" She looked up at them. He dug a mass of black vapor which it had refused to accept any. As for Mr. Swift as if it goes too high I'll warn you and you can and swallow frequently. That will make the airship was shooting upward again and just before the raid wouldn't have been instrumental in capturing the scoundrels right out of jail.

Stylistically, it's Victor Appleton meets Dylan Thomas. Technically, it's all done with trigrams.

Trigram analysis is very simple. Look at each set of three adjacent words in a document. Use the first two words of the set as a key, and remember the fact that the third word followed that key. Once you've finished, you know the list of individual words that can follow each two word sequence in the document. For example, given the input:

```
1 I wish I may I wish I might
```

You might generate:

```
1 "I wish" => ["I", "I"]
2 "wish I" => ["may", "might"]
3 "may I"  => ["wish"]
4 "I may"  => ["I"]
```

This says that the words “I wish” are twice followed by the word “I”, the words “wish I” are followed once by “may” and once by “might” and so on.

To generate new text from this analysis, choose an arbitrary word pair as a starting point. Use these to look up a random next word (using the table above) and append this new word to the text so far. This now gives you a new word pair at the end of the text, so look up a potential next word based on these. Add this to the list, and so on. In the previous example, we could start with “I may”. The only possible next word is “I”, so now we have:

```
1 I may I
```

The last two words are “may I”, so the next word is “wish”. We then look up “I wish”, and find our choice is constrained to another “I”.

```
1 I may I wish I
```

Now we look up “wish I”, and find we have a choice. Let’s choose “may”.

```
1 I may I wish I may
```

Now we’re back where we started from, with “I may.” Following the same sequence, but choosing “might” this time, we get:

```
1 I may I wish I may I wish I might
```

At this point we stop, as no sequence starts “I might.”

Given a short input text, the algorithm isn’t too interesting. Feed it a book, however, and you give it more options, so the resulting output can be surprising.

For this kata, try implementing a trigram algorithm that generates a couple of hundred words of text using a book-sized file as input. Project Gutenberg is a good source of online books (*Tom Swift and His Airship* is here). Be warned that these files have DOS line endings (carriage return followed by newline).

Objectives

Kata's are about trying something many times. In this one, what we're experimenting with is not just the code, but the heuristics of processing the text. What do we do with punctuation? Paragraphs? Do we have to implement backtracking if we chose a next word that turns out to be a dead end?

I'll fire the signal and the fun will commence…

Posted by Dave Thomas (@PragDave) Dec 16th, 2013

« Kata15: A Diversion Kata13: Counting Code Lines »

Comments

Recent Posts

- CodeKata
- CodeKata: How It Started
- Kata, Kumite, Koan, and Dreyfus
- Kata01: Supermarket Pricing
- Kata02: Karate Chop
- Kata03: How Big? How Fast?
- Kata04: Data Munging
- Kata05: Bloom Filters
- Kata06: Anagrams
- Kata07: How'd I Do?
- Kata08: Conflicting Objectives
- Kata09: Back to the Checkout
- Kata10: Hashes vs. Classes
- Kata11: Sorting It Out
- Kata12: Best Sellers
- Kata13: Counting Code Lines
- Kata14: Tom Swift Under the Milkwood
- Kata15: A Diversion
- Kata16: Business Rules
- Kata17: More Business Rules
- Kata18: Transitive Dependencies
- Kata19: Word Chains
- Kata20: Klondike
- Kata21: Simple Lists