

# Generating gobbledygook

How to make random text almost readable

## In this issue...

### ▶ WHAT'S COVERED

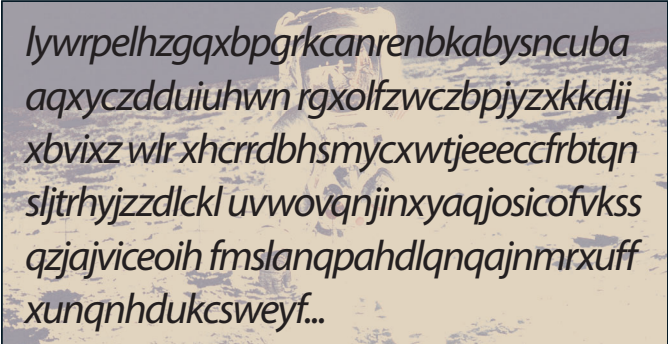
Every now and then, you need some random filler text to display in your screen or website prototype. You can use the famous 'Lorem Ipsum' text, a body of Latin much favoured by typesetters, or you can generate your own. Of course, you don't want to have it too readable, just familiar-looking. One way of generating some nonsense text is to use Markov chains.

The other day I was a little bored, so I actually scanned through the text of some of the spam email

I'd received. A lot of them were the standard rubbish. Some of them had 'appropriated' a chunk of text from an online document. And finally a small selection had some text made up of real English words, but the sentences were nonsense that almost made sense (this is known in the trade as 'word salad') until you stopped to take note of the words themselves. The main reason spammers do this is to try and beat Bayesian filters by including words that the anti-spam companies are unlikely to have flagged as likely spam content.

The first thing people do when they want to generate random text is to programmatically implement a monkey randomly bashing away at a typewriter. The process is pretty simple: in a loop, generate a random number from one to 27, output a letter from A to Z for one through 26, and a space character for 27. However, the results just aren't impressive. Figure 1 shows the result from such an algorithm.

Some of you might be thinking that a big problem with this simple algorithm is that it takes



lywrpelhzgqxbpgrkcanrenbkabysncuba  
aqxyczdduiuhwn rgxolfzwczbpjyzxkkdij  
xbvixz wlr xhcrrdbhsmycxwtjeeccfrbtqn  
sljtrhyjzdlckl uvwovqjinxyaqjosicofvkss  
qzjajviceoih fmslanqpahdlqnqajnmrxuff  
xunqnhdukcsweyf...

▲ Figure 1: Completely random text.

no account of the relative importance of each letter. In other words, since the letter E appears in English much more often than the letter Z, the text that's generated should follow the same statistical distribution as normal English prose and feature far more Es than Zs. So where can we find the formula for this statistical distribution?

We could look it up in some book or on the Internet, but the best thing to do to discover the relative distributions of the letters in English is to take a book and analyse it. I chose *The War of the Worlds* by H G Wells, mostly for fun but also because I could easily get an electronic version from the Project Gutenberg

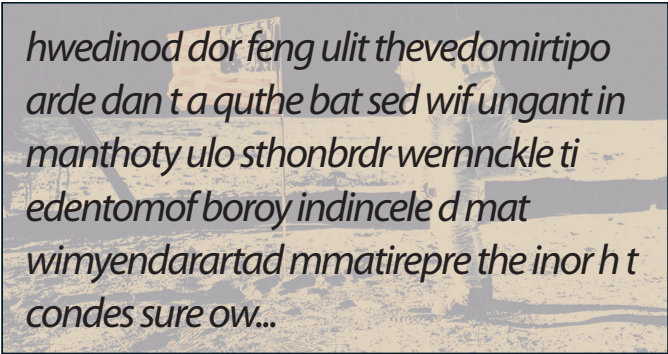
library online ([www.gutenberg.org](http://www.gutenberg.org)). The body of text being analysed is known as the 'corpus'.

### Probability problems

The analysis proceeds as follows: ignore all punctuation and count the number of times each letter and the space character appears. After that we will know how many letters plus spaces we encountered (the sum of these counts) and from that we can calculate the probability of each letter. Now that we have the probabilities of each character, we can modify the original text generator to pick letters randomly according to their relative distribution.

This is an algorithm that not many people know. Given a set of probabilities for certain events, how do you calculate a series of random events so that over a great number the observed probability of each event is the same as the required probability?

We'll assume that we have an array of probabilities for each event and that the sum of those probabilities is 1.0. Calculate a random number between 0.0 and 1.0. If this value is less than the first probability, select the first event. If not, subtract the first probability from the random value



hwedinod dor feng ulit thevedomirtipo  
arde dan t a quthe bat sed wif ungant in  
manthoty ulo sthonbrdr wernnckle ti  
edentomof boroy indincele d mat  
wimyendarartad mmatirepre the inor h t  
condes sure ow...

▲ Figure 2: Text generated based on the previous character.

(the result will be greater than or equal to zero) and repeat the check with the new value and the second probability. Eventually, you will select a particular event. Overall, over a large number of selections, you'll find that the observed distribution of events mirrors the required distribution.

Unfortunately, though, using the distribution probabilities alone still doesn't produce anything recognisable.

### The Markov method

Time to rethink our strategy a little bit. Although the latest program generates text according to the statistical distributions of the letters, English is just not like that. For example, if the previous letter were a Q, it's highly likely that the next letter is going to be a U. If the previous letter were a T, the next letter could be a vowel or an H or an R, and so on, each with a different probability. Our analysis should take into account the previous letter when generating the next random letter.

Let's try it. Go back to *The War of the Worlds* and analyse the text so that we make a note of which letters follow which letters and then generate some random text. The simplest structure to use here is a matrix with each row being the preceding character, each column being the succeeding character, and the intersection to probability of getting the succeeding character from the preceding one. This matrix will inevitably have a lot of zeros – it's known as a 'sparse matrix'.

Unfortunately, the text this new algorithm produces is still unrecognisable as English – if arguably better than some of the submissions to land on publishers' slush piles. See Figure 2 for an example, unless you've already read it, in which case

please don't write in to ask what it was supposed to mean. The only hidden message is that human writers still have some life in them yet...

I'm sure you'd recognise that, if generating a character based on the single character that occurred before is good, perhaps it would be better to use two preceding characters as a basis to deciding on the next. Or maybe three, or perhaps four.

If you've any knowledge of advanced mathematics, you might have recognised this idea as being a 'Markov chain' – and indeed, that's exactly what we are using here.

A Markov chain is a random process that moves between states in some defined set of states. Working out which state to move to is only dependent on the current state, and not on any previous state you may have already visited. In our latest example, calculating the next character is only dependent on the current character, and not on any other character we may have generated before. This type of Markov chain is known as 'order-1'. If we were to consider the previous two characters (or states) instead, we're using an order-2 Markov chain – and so on for the higher orders.

These chains are named after Andrey Markov, a Russian mathematician working in the late 19th and early 20th centuries, who specialised in probabilistic theories and calculus until his death in 1917.

The big issue we have here is that the statistical results matrix becomes extremely sparse due to the fact that the vast majority of cells have a value of zero. A better data structure to use might be a hash table, where each possible two-letter pair we see in the

*The War of the adjacent houses. I had left them nearly four weeks ago. A monstrous tripod, higher than the provision and wine shops. A couple of sturdy roughs who had gone northward. I spent that night and a special edition. Even within the shadow: greyish billowy movements, one above another, and then of the Spotted Dog had already come to hate the curate's leg, and he found one of us that are made for the last seven hours I still believed that the Heat-Ray went to and fro, and...*

▲ Figure 4: Text generated based on the previous two words.

original text is a key to a table of probabilities for the next character in the hash table.

### Refining the results

The results of order-2 Markov text generation still aren't that impressive, although we're starting to see some kind of quasi-English creeping in. It's broken, and still nonsensical, but can pass for actual language... at least at a quick glance.

Once we have the basic algorithm set up (reading and analysing the original text, building the hash table of keys and probability tables, and then using it to generate text), extending it to higher order Markov chains is fairly simple. An order-10 Markov chain (that is, calculating the next character based on the previous 10 characters) produces some pretty good nonsense English prose (see Figure 3 for the order-10 generated start to the *War of the Worlds*), but the hash table now requires a huge amount of memory to store the results of the analysis. Is there a better way to generate random text that doesn't require so much memory?

Up to now, we've been analysing the text on a character by character basis, and as we've seen, we have to ratchet quite high up the Markov order in order to

produce recognisable English words. We can't really go any higher without hitting some severe memory constraints (I tried to run order-12 and I ran out of memory for the hash table when growing it), so we should look for another approach.

Instead of using individual characters as tokens, perhaps we should use words. We could split the text up on word boundaries (the easiest definition of that is the white space between the words) and then analyse the results as we have been doing, in a standard Markov manner.

My experiments showed that a word-based order-2 Markov chain was the most efficient, both in terms of speed and memory – and it generated text that was pretty readable but complete nonsense (Figure 4 shows the result). An order-1 Markov chain produced quite unreadable nonsensical English, whereas order-3 seemed to produce huge chunks of the original text with random switches between the chunks, which didn't make it any more readable or understandable. ■

*Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express. [feedback@pcplus.co.uk](mailto:feedback@pcplus.co.uk)*

**BOOK ONE THE EVE OF THE WAR** No one would have left an abiding sense of smell, but it had a pair of very large dark eyes of a Martian from the Martians making their blue shirts, dark trousers, and singers.

▲ Figure 3: Text generated based on the previous 10 characters.