GENERATING POLITICAL NEWS USING NLTK

04 MAY 2010 | 1 COMMENT »

It’s election week here in the UK; on Thursday, we’ll be going to the polls to choose our next government. At Resolve, we’ve been doing a bunch of things related to this, including some that required us to index vast quantities of tweets and newspaper articles.

Last week I was looking at the results of this indexing, and was reminded of the fun I had playing with NLTK back in Natural Language Toolkit; as you’d expect, it has a lot of clever stuff for parsing and interpreting text. More unexpectedly (at least for me), it has the ability to take some input text, analyse it, and then generate more text in the same style. Here’s something based on the selfsame day entered Noah, and asses, flocks, and Maachah. And J said unto him, Abrah and he asses, and told all these things are against Jacob told Rachel that he hearkened not unto you. And Sarah said, I had s face of the air; for he hath broken my covenant between God and every thin creepeth upon the man: And Eber lived after he begat Salah four hundred an years, and took of every sort shalt thou be come thither.

It was the work of a moment to knock together some code that would read in all of the newspaper articles that we’d subject, run them through a Beautiful Soup-based parser to pull out the article text, and feed that into NLTK, then blog (after a little manual polishing for readability).

The result? REABot, the Resolver Electoral Analysis Robot. Here’s a sample of what I think is its finest post, which w Clegg:

They’re interested in local government, free TV licences, pension credits a trust fund, Carrousel Capital, run by local Liberal Democrats. TV Exclusive Clegg Nick Clegg, but clashed on how the vexing issue of honesty, principle policies of electric shock. It is easy to do. “Louis Vuitton advertising urged our debts”, he declared that he has delivered his strongest warning yet on first place and still obsessed with outdated class structures. Inspired by Obama’s repertoire, they advise you to send a message to voters at home. “You haven’t want to try to counter the threat of it yet,” he says.

So, what does the code look like? It’s actually trivially simple. Let’s say that we’ve downloaded all of contents of the time, with HTML-munging code here) and put them into objects with content fields. Here’s what REABot does:

```python
import nltk
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+|[^\w\s]+')

content_text = '.join(article.content for article in articles)
tokenized_content = tokenizer.tokenize(content_text)
content_model = nltk.NgramModel(3, tokenized_content)

starting_words = content_model.generate(100)[-2:]
content = content_model.generate(words_to_generate, starting_words)
print '.join(content)
```

It’s a bit of a hack — I’m sure an NLTK expert could write something much more elegant — but it works. What this, which is formed of the text of all of our relevant articles, and runs it through a tokenizer, which splits it up into words (for example) the string “I spent some time this afternoon playing with NLTK, the Python Toolkit; the book is highly recommended.” would be turned into the list ['I', 'spent', ':', 'afternoon', 'playing', 'with', 'NLTK', 'the', 'book', 'is', 'highly', 'recommended', '.']

This is then fed into an NgramModel. This is nothing to do with Science: Ngram is a word created by extension from collections of n tokens. What we’re doing with the expression nltk.NgramModel(3, tokenized_content in effect, knows about every three-token sequence (trigram) that occurs in the tokenised text [['I', 'spent', 'some', 'time'], ['some', 'time', 'this'], and so on), and knows how frequently each one occurs.

Once we’ve got the set of all possible trigrams and their respective frequencies, it’s pretty easy to see how we can generate words and a simple Markov-chain algorithm:

* Let’s say that we start off with ['The', 'tabloid'].

---

These search terms are highlighted: nltk, news generator, gilesthomas
content = content_model.generate(words_to_generate, starting_words)

It's kind of (but not really ;-) like seeding your random number generator.

And that's it! Once the text has been generated, I just copy and paste it into a Wordpress blog, do a bit of prettification (for example, remove the spaces from before punctuation and — perhaps this is cheating a little — balance brackets and quotation marks), and it takes about 5 minutes to generate an article, and to be honest I think the end result is better than a lot of the political blogs out there…

(An aside to UK readers: does anyone know if the business news in The Day Today was generated by something like this?)

CATEGORY(S): Funny, Politics, Programming, Python, Resolver Systems

ONE RESPONSE TO “GENERATING POLITICAL NEWS USING NLTK”

Dimitris Leventeas says:
5 May 2010 at 12:14 pm

Great article! It's pretty impressive how Python and NLTK made it so easy to implement this witty idea!