Creating a Custom Corpus

How to build a useable body of text

What is a corpus?

Not-so-fun fact of the day: in Latin, *corpus* means *body*. For our purposes, a **corpus** is a collection (*cough* body *cough*) of texts, on which we can perform various natural language processing (NLP) functions. In simplest terms, a corpus is a folder of text files on your computer, and corpus readers process all these text files at once, though each file can be called on individually.

NOTE: The plural of corpus is **corpora**, so be prepared to see that within this article.

Where can I find corpora?

The World Wide Web! In fact, there are tons of corpora available to download on the internet — you can find a bunch of English-language ones here. It is worth noting that a corpus can be composed of any written language, and most languages do in fact have myriad examples of corpora. *Not-so-fun fact of the day #2*: the Wikipedia corpus is composed of 1.9 billion words and is over 14GB in size.

NLP libraries (NLTK, Gensim, spaCy, etc.) come with their own corpora, though you generally have to download them separately. You

can make a corpus out of webscrapings. Or you can compile a folder of documents on your computer and turn it into a corpus. Corpora can be composed of a wide variety of file types — .yaml, .pickle, .txt, .json, .html — even within the same corpus, though one generally keeps the file types uniform.

In this article, I'll be showing you how to make a corpus out of a folder of TXT files. And since I'm hoping to someday make a corpus of my writings, most of which are in DOCX format, I'll even show you how to turn all those DOCX files into TXT in Python.

Corpora conventions

If you think you'll be using NLTK for your text processing, it's probably a good idea to follow the convention of storing your corpora in the default directory, i.e. the folder in which corpora downloaded via NLTK are stored. After installing NLTK, to find out if you have the directory, run the following:

```
import os, os.path
path = os.path.expanduser('~/nltk_data')import
nltk.data
path in nltk.data.path
```

If this returns *True*, move past this next bit; if it returns *False*, you can create the directory with the following:

```
os.mkdir(path)
```

Now go ahead a move your ***folder* of text files** — don't just throw in your text files; organization is key — into the "corpora" folder, which should be within the "nltk_data" folder. If it's not there, be sure to create that as well. You're path should now be (replace *your_corpus*):

```
'~/nltk_data/corpora/your_corpus'
```

And that's…basically it. Try calling (replace *your_corpus* and *file.txt*):

```
nltk.data.load('corpora/your_corpus/file.txt',
format='raw')
```

You should receive a bytes object.

What's a bytes object? you may ask. Bytes objects are encoded strings that can be stored directly on your computer's hard drive. Strings need to be encoded (more on that later) before your computer can read them. A bytes object can be read by machines, whereas a string can be read by humans, i.e. you (presumably).

Setting the format to 'raw' is convention, but you can also set it to 'text' to return a string. If your file is .pickle, .yaml, or .json, no format parameter is required. NLTK knows (almost) all.

Important note: while there may be some increased functionality by placing your custom corpus in the *nltk_data/corpora* folder, I've found no real need to do so. Chalk it up to convention and organization. If it makes more sense for you to have your corpus in your project folder, go for it.

So now what?

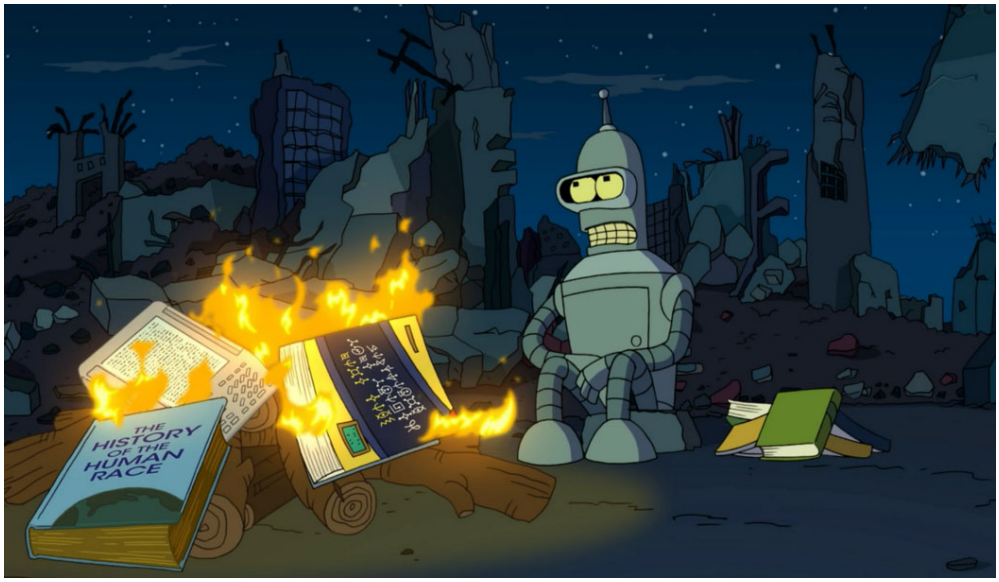
Alright, so that was pretty easy, and you may be wondering what the heck a corpus is good for. Lots of stuff! Try running (again, replace *your_corpus* and possibly *latin1*, depending on your files' encoding):

```
corpus_root =
os.path.expanduser('~/nltk_data/corpora/your_corpus')
corpus = PlaintextCorpusReader(corpus_root, '.*',
encoding='latin1')
```

```
corpus.words()
# corpus.sents()
# corpus.paras()
```

Thanks to the handy `.*` — which accesses all the files in your root path — you should get a list of all the words in your corpus folder. That's right! Each of your files has been processed as one entity. It has been turned into one list of all the words they contain, as well as a list of sentences and paragraphs (use the commented code for these). From there the possibilities are pretty much endless. You can analyze; you can predict; you can one-hot-encode; you can turn it into word vectors; you can generate text; you can do too much to address here.

The frustrations of encoding



Character encoding can be pretty weird. My original TXT files were encoded in Latin-1 (aka *iso-8859–1*) and my apostrophes turned into this strange character — 'Õ' — when accessing my corpus. Even though that's a fairly simple fix in Python, in the future I'll be sure to encode in or convert to UTF-8.

UTF-8 is capable of encoding the over one million characters that make up Unicode. It is almost definitely your best option and should not lead to strange interactions like the case above. You can read more on Unicode here.

Knowing how your TXT files are encoded is important. To find out how a file is encoded, you can use the following code in Terminal (again, replace *your_file.txt*):

```
file -I your_file.txt
```

You should get something like:
your_file.txt: text/plain; charset=iso-8859–1

IMPORTANT: To reiterate, use UTF-8 encoding if you have the option, and/or consider converting your TXT files to that. Luckily, saving a TXT file via Python defaults to encoding in UTF-8, so use that to your advantage.

Bonus: easily convert your DOCX files to TXT

If you're like me, most of the files you may want to corpustulate® are DOCX files, which aren't readable by Python without the proper libraries. With one such library, docx2txt, you can easily read DOCX files and then save the string as TXT. After installing the library, run the following code:

```
import docx2txtresult =
docx2txt.process('path_to_file/your_file.docx')with
open('desired_path/your_file.txt', 'w') as f:
    f.write(result)
```

And there you have it — a process that you can easily loop over to turn a folder of DOCXs into TXTs.

Up next: A beginner's attempt at a model-based poetry generator.

A note from Plain English

Did you know that we have launched a YouTube channel? Every video we make will aim to teach you something new. Check us out by clicking here, and be sure to subscribe to the channel 😎

76

2