This pair of Python3 programs makes it possible to perform the kind of stylometric analyses of text files pioneered by John Burrows and associates (Burrows, 1987, 1992; Burrows & Craig, 2001).

Why I Wrote this Software
I wrote an early version of these programs in Snobol (Spitbol implementation) in 1994. Then in 2007 I developed slightly enhanced versions in Python2. I stopped using Python2 in 2010, so I have been meaning to upgrade them for a while, and was prompted to do so in 2015 by a user request. The recent Python3 edition is, I believe, an improvement on its predecessors.

The main point of the software is to enable a Burrows-inspired approach to document analysis. In a nutshell, the first program (dox2vox.py) reads a corpus of text files and produces from it a vocabulary listing. The second program (vox2dat.py) takes in a vocabulary listing such as produced by dox2vox.py as well as reading the same or another corpus and produces a data file in a format that can conveniently be imported into R (R Core Team, 2013) for further processing. Each row of that data file describes a text document from the input corpus. The columns give scores for each text on a number of measures. The first few columns are basic housekeeping data; the next 10 (which may be added to) are a variety of vocabulary-richness scores; the rest are percentage occurrence rates of the wordforms in the input vocabulary file. The number of wordforms whose occurrence rates will be written to file is chosen by the user.

A major reason for splitting the overall task into two is that it gives a user the chance to hand-edit the vocabulary file produced by dox2vox.py before it is used by vox2dat.py. It also permits one corpus to be analyzed using the vocabulary of another corpus.

In summary, vocsoft is a front-end for analyses to be carried out in R (or similar systems), such as Principal Components Analysis, Cluster Analysis, Discriminant Analysis and suchlike. I find this division of labour convenient since, in my view, Python3 is better for text and string processing whilst R is more suited to statistical calculations.

Setting Up
First you need Python3. If you don't have it already, the latest version can be downloaded and installed from the Python website: www.python.org. This is usually quite straightforward. The only snag is if you have Python2 and want to keep using it. Then you'll probably have to set up a specific command to run whichever version you use less frequently.

Next step is to unpack the vocsoft.zip file. After unpacking it (into a top-level folder called "vocsoft", unless you want to do quite a lot of editing), you should find the following subfolders.
The programs are in p3. Sample corpora for testing will be found in samples. Subfolder op is the default location for output files and parapath is a convenient place for storing parameter files, which will be explained later. In Windows, it is most convenient to install vocsoft at the top level of the C:\ drive, at least to start with.

**Corpus Format**

Vocsoft is a document-oriented system. Thus an input corpus consists of a number of text files (in UTF8 encoding). Each file is treated as an individual document. Ideally each file should contain running text without markup. Markup (e.g. HTML, SGML & suchlike) is not handled well, so running these programs on marked-up documents will usually give strange results.

In the samples folder you will find 6 subfolders (bottlabs, britfict, cics, ew, feds and tedtrans). These contain data sets that enable you to start using the system, prior to collecting &/or reformatting your own corpora.

The first, bottlabs, contains a small corpus of back-label texts from beverage bottles, mostly beer and wine.

The second, britfict, contains 36 fictional texts written by 12 different British authors. Most are complete novels, though three are chapters or sections from larger works.

The second contains writings by several Latin authors, the three main ones being: Marcus Tullius Cicero, the famous Roman orator, Mark-Antoine Muret, known as Muretus, and Carlo Sigonio. This dataset arises from an interesting authorship problem. Background information can be found in Forsyth et al. (1999), but in a nutshell the problem revolves around a work called the *Consolatio* which Cicero wrote in 45 BC. This was thought to have been lost until in 1583 AD when Carlo Sigonio claimed to have rediscovered it. He died the following year never having made public the manuscript, but published a printed version in Venice with himself named as editor. Scholars have argued since then over whether the book is genuinely a rediscovery of Cicero's lost work or a renaissance fake.

The subfolder ew contains 46 short stories by Edith Wharton as well as 6 chapters from her novels and some comparison texts by Henry James and Marion Mainwaring. This corpus is interesting because when Edith Wharton died in 1937 she left her novel *The Buccaneers* unfinished. It was later completed by Marion Mainwaring in 1993. Two chapters by Wharton as well as 2 by Mainwaring are included in the sample on disk.

The feds subfolder contains writings by Alexander Hamilton and James Madison, as well as some contemporaries of theirs. This is related to another notable authorship dispute, concerning the *Federalist Papers*, which were published in New York in 1788. Of the 85
essays in that book, 51 are known to have been written by Hamilton, 14 by Madison, 5 by John Jay and 3 jointly by Hamilton and Madison together. That left 12 disputed papers (numbers 49-58 and 62-63) claimed by both Hamilton and Madison. For more background see Holmes & Forsyth (1995).

The tedtrans subfolder contains 1555 transcripts of talks, in English, given under the TED.com initiative. Obtained from collection held at WIT3 website https://wit3.fbk.eu.

Running DOX2VOX
This program reads in a corpus of texts and produces a frequency-ordered list of vocabulary items in 2 versions, one machine-readable (for input to vox2dat.py) and the other intended to be more readable by humans. When you run it, it will ask you to type in a jobname. This should be an alphanumeric string which will be used to link together the output files produced so that they can be seen to be part of the same project. It can also be used as a way of supplying nonstandard parameter settings to the program, as will be explained below.

You should see on screen something like the listing below, which comes from a run using the britfict corpus as input. In this example the program is executed at the command line from another working directory (c:\2015\) which means that the full path of the program has to be given.

c:\2015>python c:\vocsoft\p3\dox2vox.py
C:\vocsoft\p3\dox2vox.py 1.4 Fri Nov 13 16:41:42 2015
command-line args. = 1
progpath : C:\vocsoft\p3
working folder:  C:\2015
please give jobname : ew
ew to be used as jobname.
jobname [ew] :
atomize [1] :
casefold [1] :
docpaths : c:\vocsoft\samples\britfict\novs
docpaths = c:\vocsoft\samples\britfict\novs
filetail [.txt] :
outpath [C:\vocsoft\op] :
snipsize [115] : 1024
snipsize = 1024
topvocs [144] :
vocdump [ew_vocs.dat] :
vocfile [ew_vocs.txt] :
wordonly [1] :
please choose sortcol
0 corprate
1 docrate
2 sniprate
3 textmean
4 textmid
Give option number: 2
Mode 2 chosen : sniprate
sortcol = sniprate
c:\vocsoft\samples\britfict\novs\files found on c:\vocsoft\samples\britfict\novs\ = 36
texts read from c:\vocsoft\samples\britfict\novs\ = 36
total word tokens = 7140077
total vocabulary size = 60503
tokens occurring at least 3 times = 32514
wordform lines = 144
User inputs required here have been marked in **bold face**, i.e. the program launch command, the jobname, the document input path specification, the nonstandard value for the parameter 'snipsize' and the choice of 'sniprate' as the sorting criterion. The effect of the parameters is described in the next section.

Notice that in the listing above the portion from "please give jobname" to "Give option number" is where the user gives input values for a number of program parameters. Where the program already has computed a default value, there is an item within square brackets indicating that value. In such cases just pressing the "Enter" key (= "hitting Return") will select that default value. The idea is to save typing. Where the user gives an input other than hitting return, i.e. overrides the default, the value given is echoed after an equal-sign. These user-input conventions are also used by vox2dat.py.

**Interpreting the output listing of dox2vox.py**

The listing below shows the output file `ew_vocs.txt` derived from the run above. This is the output intended for human inspection.

```
\ ` rank corpfreq docfreq snipfreq corprate corpsum docrate sniprate textmean textmid
the 1 299673 36 6967 4.20 4.20 100.00 100.00 4.50 4.28
and 2 231541 36 6967 3.24 7.44 100.00 100.00 3.26 3.20
to 3 226027 36 6967 3.17 10.61 100.00 100.00 3.02 2.99
of 4 182590 36 6967 2.56 13.16 100.00 100.00 2.63 2.48
a 5 141935 36 6967 2.05 15.21 100.00 100.00 2.10 2.16
in 6 111386 36 6967 1.59 16.80 100.00 100.00 1.62 1.63
that 7 98096 36 6967 1.37 18.17 100.00 100.00 1.33 1.18
with 8 61841 36 6965 0.87 19.04 100.00 99.97 0.87 0.88
as 9 70781 36 6964 0.99 20.03 100.00 99.96 0.96 0.95
it 10 84803 36 6963 1.19 21.22 100.00 99.94 1.24 1.17
for 11 61993 36 6960 0.87 22.09 100.00 99.90 0.84 0.79
but 12 51499 36 6951 0.72 22.81 100.00 99.77 0.70 0.67
not 13 61663 36 6943 0.86 23.67 100.00 99.66 0.83 0.80
be 14 57439 36 6928 0.80 24.48 100.00 99.44 0.73 0.67
at 15 45193 36 6920 0.63 25.11 100.00 99.45 0.65 0.64
was 16 85962 36 6878 1.20 26.31 100.00 98.72 1.29 1.40
have 17 49223 36 6852 0.69 27.00 100.00 98.25 0.60 0.49
had 18 61660 36 6845 0.86 27.87 100.00 98.25 0.90 0.89
by 19 31121 36 6804 0.44 28.30 100.00 97.66 0.44 0.43
on 20 35493 36 6781 0.50 28.80 100.00 97.33 0.51 0.54
all 21 30462 36 6776 0.43 29.23 100.00 97.26 0.42 0.42
so 22 35141 36 6774 0.49 29.72 100.00 97.23 0.47 0.48
this 23 33143 36 6765 0.46 30.18 100.00 97.10 0.45 0.39
i 24 167132 36 6736 2.34 32.52 100.00 96.68 2.21 1.95
his 25 71990 36 6709 1.01 33.53 100.00 96.30 1.06 1.04
he 26 91701 36 6701 1.28 34.82 100.00 96.18 1.33 1.24
from 27 25729 36 6679 0.36 35.18 100.00 95.87 0.36 0.36
is 28 45972 36 6619 0.64 35.82 100.00 95.01 0.60 0.59
which 29 33017 36 6614 0.46 36.28 100.00 94.93 0.47 0.41
no 30 24954 36 6609 0.35 36.63 100.00 94.86 0.36 0.37
if 31 27596 36 6553 0.39 37.02 100.00 94.06 0.36 0.36
would 32 28948 36 6522 0.41 37.42 100.00 93.61 0.39 0.37
when 33 21937 36 6477 0.31 37.73 100.00 92.97 0.30 0.30
one 34 20475 36 6469 0.29 38.02 100.00 92.85 0.29 0.27
what 35 25287 36 6427 0.35 38.37 100.00 92.72 0.33 0.33
an 36 20948 36 6424 0.29 38.66 100.00 92.21 0.30 0.29
him 37 39658 36 6394 0.56 39.22 100.00 91.78 0.58 0.57
you 38 83134 36 6379 1.16 40.38 100.00 91.56 1.10 1.10
or 39 22949 36 6369 0.32 40.70 100.00 91.42 0.32 0.31
been 40 21743 36 6292 0.30 41.01 100.00 90.31 0.30 0.28
my 41 65566 36 6286 0.92 41.92 100.00 90.23 0.91 0.57
her 42 79189 35 6217 1.11 43.03 97.22 89.23 1.04 0.77
were 43 21244 36 6188 0.30 43.33 100.00 88.82 0.32 0.33
very 44 19957 36 6180 0.28 43.61 100.00 88.70 0.29 0.25
there 45 19440 36 6139 0.27 43.88 100.00 88.12 0.29 0.28
more 46 16805 36 6122 0.24 44.12 100.00 87.87 0.23 0.22
me 47 49397 36 6120 0.69 44.81 100.00 87.84 0.61 0.48
said 48 35188 36 6059 0.49 45.30 100.00 86.97 0.50 0.44
who 49 19482 36 6007 0.27 45.58 100.00 86.22 0.27 0.24
could 50 17983 36 6007 0.25 45.83 100.00 86.22 0.27 0.25
\`
In this output the most frequent 144 wordforms in the input corpus have been listed in descending order (followed by a dump of the program's parameter settings, which can be very useful for checking purposes: see Appendix). The question is: what does "most frequent" mean? In this case the ordering is determined by 'sniprate'. This is computed as the percentage of snippets in which the wordform occurs. A snippet is a block of text of a fixed size, given by the 'snipsize' parameter, which was set to 1024 above. The normal default for snipsize is 115, the number of words in Shakespeare's 18th sonnet, but with full-length novels such as in the britfict sample, that leads to rather too many snippets for convenience.

From the output above, it can be seen that the first 7 words, from 'the' to 'that' occur in 100% of the snippets in this corpus. Even the 144th item by rank, 'away', occurs in almost half of the snippets (48.66%). The options for frequency ordering are as follows.

<table>
<thead>
<tr>
<th>ordering criterion</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>corprate</td>
<td>This is simply the relative frequency, expressed as a percentage, in the corpus as a whole, i.e. the total number of occurrences of the wordform divided by the total number of tokens in the corpus (multiplied by 100). It is the default value if none other is given; and it is what is generally meant by loose usage of the term 'frequency'.</td>
</tr>
<tr>
<td>docrate</td>
<td>This is the percentage of documents in which the wordform occurs. With large documents, as in the case above, many common words will have a docrate of 100 percent.</td>
</tr>
<tr>
<td>sniprate</td>
<td>This is the percentage of 'snippets' (of size given by 'snipsize') in which the wordform is found.</td>
</tr>
</tbody>
</table>
This is computed by calculating the percentage occurrence rate of the wordform in every input document and then calculating the mean (arithmetic average) of those rates.

This is computed by calculating the percentage occurrence rate of the wordform in every input document and then calculating the median of those rates.

Most of the vocabulary items in a listing such as that above are frequent function words that would be considered "stop words" in the context of Information Retrieval. Such words have proved very useful in stylometry, particularly as authorial markers (e.g. Mosteller & Wallace, 1984; Burrows, 1992; Holmes, 1994).

It can be seen from this listing that the order determined by 'sniprate' is not the same as what would be imposed by using any of the other sorting columns. For example, the word "i" (mostly the first-person pronoun "I", probably with a few instances of the Roman numeral I confounded by case folding (implied by 'wordonly' being equal to 1)) which appears at rank 24 would come fifth in order on the basis of 'corprate'. The finding that pronouns are more variable in their distribution is general across many text types.

The machine-readable (.dat) output file

The dox2vox.py program also dumps the selected vocabulary in a tab-delimited format designed to be read by vox2dat.py (though it can also be easily imported into R). An abbreviated example, from the run above, is shown below.

<table>
<thead>
<tr>
<th>wordform</th>
<th>rank</th>
<th>corpfreq</th>
<th>docfreq</th>
<th>snipfreq</th>
<th>corprate</th>
<th>corpusum</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>1</td>
<td>299673</td>
<td>6967</td>
<td>4.20</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>and</td>
<td>2</td>
<td>231541</td>
<td>6967</td>
<td>3.24</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>226027</td>
<td>6967</td>
<td>3.17</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>of</td>
<td>4</td>
<td>182590</td>
<td>6967</td>
<td>2.05</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>146385</td>
<td>6967</td>
<td>1.59</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>in</td>
<td>6</td>
<td>113186</td>
<td>6967</td>
<td>1.37</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>that</td>
<td>7</td>
<td>98096</td>
<td>6967</td>
<td>1.23</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>with</td>
<td>8</td>
<td>61841</td>
<td>6965</td>
<td>0.87</td>
<td>100.00</td>
<td>99.97</td>
</tr>
<tr>
<td>as</td>
<td>9</td>
<td>70781</td>
<td>6964</td>
<td>0.99</td>
<td>100.00</td>
<td>99.96</td>
</tr>
<tr>
<td>it</td>
<td>10</td>
<td>84883</td>
<td>6963</td>
<td>1.19</td>
<td>100.00</td>
<td>99.94</td>
</tr>
</tbody>
</table>

Here only the first and last 10 items have been retained.

All the various frequency measures are included, but the only column that is read by vox2dat.py is the first. Thus, if you wish to insert a word into the vocabulary to be used by vox2dat.py, you don't have to compute any associated statistics. All that is needed is a line beginning with that word. An example of an external vocabulary file is cobuild.vox which is
included in the samples folder. This lists the 111 most frequent word tokens in the Cobuild corpus from a time when that corpus contained approximately 7 million words.

**Executing VOX2DAT**

An example of the kind of screen output resulting from running vox2dat.py (this time from within the IDLE environment) is shown below. In this case the ew.txt file in the parapath subdirectory containing the following lines

```#
# ew parameter file :
docpaths c:\vocsoft\samples\britfict\novs
folders c:\vocsoft\samples\ew\taletext, c:\vocsoft\samples\ew\holdout2
wordonly 1
```

was used; thus the 'folders' question, below, was answered simply by hitting return, selecting the 2 input folders indicated in the parameter file.

```>>>
C:\vocsoft\p3\vox2dat.py 1.4 Fri Nov 13 16:48:37 2015
command-line args. = 1
progpath : C:\vocsoft\p3
working folder: C:\vocsoft\p3
please give jobname : ew
ew to be used as jobname. ['C:\\vocsoft\\p3', 'C:\\vocsoft\\parapath', 'C:\\vocsoft']
atomize [1] :
casefold [1] :
datfile [ew_vars.dat] :
filetail [.txt] :
folders [c:\vocsoft\samples\ew\taletext, c:\vocsoft\samples\ew\holdout2] :
outpath [C:\vocsoft\op] :

snipsize [115] : 1024

snipsize = 1024
topvocs [144] :
voxfile [ew_vocs.dat] :
wordonly [1] :
c:\vocsoft\samples\ew\taletext
c:\vocsoft\samples\ew\holdout2
vocabulary items read from ew_vocs.dat = 144
files found on c:\vocsoft\samples\ew\taletext = 44
files found on c:\vocsoft\samples\ew\holdout2 = 13
texts read from c:\vocsoft\samples\ew\taletext, c:\vocsoft\samples\ew\holdout2 = 57
output lines = 57
data values listed on C:\vocsoft\op\ew_vars.dat
C:\vocsoft\p3\vox2dat.py done on Fri Nov 13 16:48:58 2015
after 20.3908069 seconds.
``` 

It should be noted that the program has read its vocabulary from the file ew_vocs.dat, in other words it expected a file such as produced by dox2vox.py with a name composed of the jobname with "_vocs.dat" appended. It should also be noted that this input file was derived from the britfict corpus, but the documents scanned by vox2dat.py are from c:\vocsoft\samples\ew\taletext and c:\vocsoft\samples\ew\holdout2, as indicated following the 'folders' question above. In other words, writings by Edith Wharton (and a couple of relevant comparison authors) are to be analyzed using what might be regarded as a generic vocabulary of British fiction.

The data grid produced is written onto the file ew_vars.dat. Only the header and first line of this file are listed below, since its rows have 160 elements so the lines are too wide to...
Information concerning the output variables is given in the table below.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepath</td>
<td>This is the directory path of the file concerned, up to but excluding the filename itself.</td>
</tr>
<tr>
<td>textname</td>
<td>This is the name of the text file concerned, without its directory path prefix.</td>
</tr>
<tr>
<td>filenum</td>
<td>This is a serial number, giving the order in which the files were processed.</td>
</tr>
<tr>
<td>totchars</td>
<td>This is the total number of characters in the file.</td>
</tr>
<tr>
<td>tottoks</td>
<td>This is the total number of tokens (which might not always be words) in the file as computed by the program's tokenizer.</td>
</tr>
<tr>
<td>totvocs</td>
<td>This is the total number of distinct tokens found in the file, i.e. the vocabulary size.</td>
</tr>
</tbody>
</table>
This is Brunet's W (Brunet, 1978), a measure of vocabulary richness computed as
\[ W = N \cdot (V \cdot (-0.169)) \]
where N is tottoks and V is totvocs and the circumflex signifies raising to the power. (This measure is actually lower with a richer vocabulary.)

This is Simpson's index of diversity (see Upton & Cook, 2006)
\[ S = 1 - \sum(p_j^2) \]
where each \( p_j \) is the proportion of token j in the overall total -- with the modification that *Hapax Legomena* are excluded from the computation. Unfortunately it is somewhat correlated with text length.

This is the number of *Hapax Legomena* (once-occurring tokens) in the text divide by tottoks. It also is unstable across texts of different lengths.

This is the number of *Dislegomena* (twice-occurring tokens) divided by \( V \), the number of distinct words in the file, an index proposed by Sichel (1975).

This gives the mean (average) number of *Hapax Legomena* in each snippet of the file, divided into snippets of size equal to snipsize, expressed as a percentage.

This gives the standard deviation of the values used to compute sniphaps.

This gives the mean type-token ratio as a percentage, 100*V/N, of all the snippets in the file. Unlike overall TTR, this can be used as a vocabulary-richness measure among texts of different lengths (provided they are long enough to contain more than a handful of snippets).

This gives the standard deviation of the values used in computing snipttr.

The remaining columns give the relative frequencies, expressed as percentages, of the vocabulary items read in for each text. Short tokens of less than four characters have an underscore appended, e.g. 'by_', to avoid confusion with reserved words in packages such as SPSS. Tokens that aren't entirely alphabetic have a prefix 'v_' added.

**What can be done with this kind of output?**
The data grid produced is merely a means to an end. The main idea is that it will be read into R or another statistical system for further processing. For example, the R command

`ew = read.delim("c:\vocsoft\op\ew_vars.dat")`
would read the data file created above into a data-frame called ew with 57 rows and 160 columns. Incidentally, the first 52 rows of this file refer to writings by Edith Wharton and the last five by comparison authors. These five include two chapters by Henry James (a mentor of Wharton) and two by Marion Mainwaring, from the portion she added to complete *The Buccaneers*, the novel left unfinished by Wharton. There is also an English translation, made in 1968, of *Les Metteurs* which Wharton wrote in French but never translated herself.

A scatter plot produced in R of these 57 texts, where the axes are the values of snipttr and the percentage occurrence rate of the word 'where', is shown below.

![Scatter plot](image)

This shows that Marion Mainwaring's additions to *The Buccaneers* (the 2 green points marked MM) had a higher snippet-based type-token ratio than all but 1 of Edith Wharton's texts. This suggest she used a more varied vocabulary. In addition, Mainwaring used the word 'where' with higher frequency than in any text in the Wharton sample. Thus her chapters appear as outliers on this plot. The 2 novel chapters by Henry James straddle the main group of Wharton's works, indicating that they wouldn't be typical of Wharton, though this pair of variables would not be adequate to distinguish these 2 authors. The translation from French, by Nolan, is, like Mainwaring's chapters, higher on snipttr than all but 2 of the 57 texts in this sample. This proves nothing in itself, but it does suggest that the hypothesis
that translation, by forcing a translator to think of alternative expressions, may tend to increase vocabulary richness.

The next two example graphs illustrate another way of looking at this sort of data. In these, the usage rates in the 52 texts by Edith Wharton have been plotted against calendar year. The dates concerned can be found in the file ewdates.dat in the ew subfolder of the samples on disk. They are, as far as can be ascertained, the composition years of the texts concerned. Whereas her rate of usage of 'upon' declined, especially after 1905, her usage rate of 'well' increased as she grew older.
Another example is illustrated below. This arises from the bottlabs corpus, a collection of short texts taken from the back-labels of beverage bottles. In this case, after creating a vocabulary of 144 items with dox2vox.py and then creating a data grid with vox2dat.py, the resultant data grid was read into R and subjected to a Principal Components Analysis (PCA) using the prcomp() built-in function.

The graph shows each of the file names (truncated to 8 characters) positioned in the space of the first 2 principal components, which account for 22% of the overall variance. Beers are in black, wines are coloured blue, soft drinks are in green and the sole cider is coloured red. The main categories, beer and wine, are fairly well separated on these 2 dimensions, with the soft drinks occupying an intermediate position. Unexpectedly, there seems to be evidence for 2 distinct subclasses among the beers, separated vertically on the second dimension above and below the zero point. I haven’t yet worked out what factors distinguish these subgroups. Initial suspicion is that it depends on whether their labels include a standard warning from the UK Chief Medical Officers concerning alcohol intake.

Finally, the last plot deals with some of the vocabulary richness measures computed by vox2dat.py. It is derived from running vox2dat.py on the transcripts in the tedtrans folder. On a single graph it shows how three indices vary with text length -- uncorrected Type-Token Ratios (TTRs) in black, Herdan’s C in red and snipttr (divided by 100 to fit on the same scale) in green. The lowess() smoothing function of R has been used to overlay smoothed lines on the points of Herdan’s C and snipttr. (TTR is obviously size-dependent to the naked eye, so no line is overlaid on those points.) It is evident that Herdan’s C does decline with text size, although less markedly than raw TTR. Clearly, snipttr is much less strongly associated with number of tokens, though even here a slight downward trend is discernible.
Acknowledgements

Texts from the TED initiative (www.ted.com) obtained from https://wit3.fbk.eu with thanks.

Thank you for reading this far. :-)

References


Appendix: Parameter Files

These programs have a number of parameter values that can be set to influence their behaviour. On the whole it is intended that most of them have sensible values predefined, so they can safely be ignored by a user. However, some can't be pre-determined and others can usefully be altered for particular purposes.

There is a sequence in which vocsoft programs determine the values of their adjustable parameters, which defines a priority ordering, as follows. If present, values given in later steps over-ride those from previous steps.

1. Each program has an initial list of built-in default values.
2. Each program searches for a file called settings.txt in the same directory as the program itself; if found, this is read and each line scanned for a parameter name (e.g. 'outpath') followed by 1 or more blank spaces followed by at least 1 nonblank. Anything after the blanks will be treated as a new value for that parameter (subject to certain minimum and maximum limits imposed on some numeric values).
3. Each program will look through the following directories (if they exist) in the following order: the current working directory, the "parapath" directory of the parent of the current working directory and the parent directory of the current working directory; e.g. if the working directory is "c:\vocsoft\p3" the order of search will be as follows
   ['C:\vocsoft\p3', 'C:\vocsoft\parapath', 'C:\vocsoft']
   stopping when a file with the name of the jobname followed by ".txt" is found. If found, that file will be read for parameter values in the manner described in (2) above, overwriting any previously stored values.
4. Finally, the programs will ask the user for values of a subset of adjustable of parameter, using in each case as default the value resulting from steps (1)-(3) above. These are presented within square brackets. If the user just hits Return, the default is accepted. In most cases, this minimizes typing.

However, some parameter values, such as input folders, cannot be guessed in advance, and some may be unsuitable in a particular experiment. In such cases, it usually saves work to prepare a parameter file, using a text editor such as Notepad or Notepad++, freely available at https://notepad-plus-plus.org/download/v6.7.4.html which should be named with the jobname followed by ".txt".

For example, the parameter file ew.txt on folder vocsoft\parapath contains the following lines.

```markdown
## ew parameter file :
docpaths c:\vocsoft\samples\britfict\novs
## docpaths c:\vocsoft\samples\ew\taletext
folders c:\vocsoft\samples\ew\taletext, c:\vocsoft\samples\ew\holdout2
wordonly 1
```

The first line, beginning with "##", is in effect a comment, since "##" isn't recognized as a parameter name. The next line specifies where to seek the files from which dox2vox.py will build its vocabulary. The parameter name 'docpaths' is plural because this parameter can have a list of directory names, separated by commas; though in the present example, all input files reside in a single folder, containing the sample of British fiction.

The third line is another comment, but it can easily be edited, by deleting the leading "##", to select a different source (Edith Wharton's stories) for a comparison experiment.

The line starting with "folders" will be read only by vox2dat.py, as specifying a comma-separated list
of folders where the texts to be processed reside. You can see that editing this in a text file is likely to be easier than (repeatedly) entering it interactively. (Likewise with 'docpaths'.)

Finally, the line setting the 'wordonly' parameter to 1 is redundant, unless the settings.txt file has been corrupted, since this is the value set in that file. It means that only tokens starting with an alphanumeric character will be counted.

As an introduction, the parameter dump at the end of the file ew_vocs.txt is reproduced below, with explanations inserted after those parameters which are of most interest to a user.

Parameter settings:

atomize 1
When set to 1 this means that the internal tokenizer will be used to split the input texts into tokens (not always lexical words). Only set this to 0 if your texts have been tokenized already, with whitespace as inter-token separation.

casefold 1
This instructs the program to convert all upper-case letters to lower case. Set this to 0 if you wish to retain upper/lower-case distinction.

dateline Sat Sep 12 15:58:56 2015

doclist <class 'list'> of 36 items.
docpath c:\vocsoft\samples\britfict\novs\
docpaths c:\vocsoft\samples\britfict\novs\This is the comma-separated list of folders containing the input texts to be read. (The singular 'docpath' is an internal value used only within the program. Equivalent to 'docpaths' with vox2dat.py is 'folders'.)
docs 36
This isn't settable by a user, but it is useful to know that it counts the number of documents processed.
dumpname C:\op\ew_vocs.dat
dfiletail .txt
If this parameter has a non-empty string value, only files that end with this string will be read.
id c:\vocsoft\p3\dox2vox.py
jobname ew
This is the jobname, which is used to link related outputs.

minfreq 3
This is an internal value: only tokens that occur more than twice overall will be considered for inclusion in the output vocabulary.

outpath C:\op\
This specifies the directory path where output files will be written.

pathlist ['c:\vocsoft\samples\britfict\novs\']
progname c:\vocsoft\p3\dox2vox.py
propath c:\vocsoft\p3\
punxtab <class 'dict'> of 25 items.

snipsize 1024
This specifies the size (number of tokens) in a snippet, for the purpose of calculating snippet-based scores. Standard snippet size is 115 tokens.

sortcol sniprate
This defines which column will be used for ordering the output. Default is 'corporate'. (Table on page 6 describes the options.)
topvocs 144
This specifies the number of high-frequency tokens to be included in the vocabulary. The default value is 144.
totsnips 6967
The program computes the number of snippets and lists this number for information.
tottoks 7140077
This is computed by the program. Here we find that the input corpus contains over 7 million alphanumeric tokens (mostly lexical words, but probably also including some numbers).

vocdump  ew_vocs.dat
Machine-readable file of output vocabulary. Only file name needed since it will always be written on the outpath folder.

vocfile  ew_vocs.txt
If not specified, this will be formed from the jobname with ".vocs.dat" appended, and saved in the outpath folder.

vocsize  60503
This is the number of distinct vocabulary items, computed by the program. (Called totvocs by vox2dat.py.)

voutname  C:\op\ew_vocs.txt
whereat  C:\2015\-

The program shows the working directory from which the program was executed.
wordonly  1
When equal to 1 this specifies that only tokens starting with an alphanumeric character will be considered; if it is 0, all tokens, including punctuation, will be included in the vocabulary.

It should be noted that the list of input paths to dox2vox.py is 'docpaths', while the list of input paths to vox2dat.py is called 'folders'. This allows the same parameter file to be used with both programs, even when the vocabulary generated from one corpus is tested on a different corpus. Likewise, vocdump is the parameter to be used if its required to give a nonstandard file name to dox2vox.py for the machine-readable vocabulary output (no path specification needed as it is always written to the outpath folder), whereas voxfile is the parameter to tell vox2dat.py to read from a nonstandard vocabulary file (full path specification required). Different names are used to avoid accidentally overwriting a useful word list.

N.B. At present, if you have more than 1 blank line in a parameter file, the input routine chokes. I do plan to fix this at some stage; in the mean time, it is quite easy to delete empty lines using a text editor.