VOCSOFT Stylometric Software

(User Notes by Richard Forsyth, November 2015)

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.

op
p3
parapath
samples

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 occuring at least 3 times = 32514
wordform lines = 144
```

```
vocabulary listing on C:\vocsoft\op\ew_vocs.txt
data values listed on C:\vocsoft\op\ew_vocs.dat
C:\vocsoft\p3\dox2vox.py done on Fri Nov 13 16:43:02 2015
after 79.8477452 seconds.
```

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	146385	36	6967	2.05	15.21	100.00	100.00	2.10	2.16
in	6	113186	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	1.30	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	84883	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	99.33	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	98.35	0.64	0.62
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	25027	36	6460	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	65568	36	6286	0.92	41.92	100.00	90.23	0.81	0.57
her	42	79189	35	6217	1.11	43.03	97.22	89.23	1.04	1.07
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

than	51	14968	36	5914	0.21	46.04	100.00	84.89	0.21	0.19
now	52	15575	36	5905	0.22	46.26	100.00	84.76	0.21	0.22
she	53	59181	36	5879	0.83	47.08	100.00	84.38	0.79	0.80
out	54	15837	36	5876	0.22	47.31	100.00	84.34	0.23	0.22
they	55	20608	36	5854	0.29	47.60	100.00	84.02	0.30	0.28
do	56	18050	36	5800	0.25	47.85	100.00	83.25	0.25	0.25
any	57	15054	36	5749	0.21	48.06	100.00	82.52	0.21	0.19
will	58	24949	36	5672	0.35	48.41	100.00	81.41	0.28	0.25
up	59	14294	36	5647	0.20	48.61	100.00	81.05	0.21	0.20
them	60	16161	36	5631	0.23	48.83	100.00	80.82	0.24	0.21
are	61	17972	36	5630	0.25	49.09	100.00	80.81	0.24	0.23
should	62	15394	36	5606	0.22	49.30	100.00	80.47	0.20	0.20
into	63	12599	36	5553	0.18	49.48	100.00	79.70	0.18	0.18
much	64	12783	36	5533	0.18	49.66	100.00	79.42	0.18	0.17
then	65	13541	36	5525	0.19	49.85	100.00	79.30	0.19	0.18
such	66	13783	36	5501	0.19	50.04	100.00	78.96	0.18	0.16
little	67	14082	36	5489	0.20	50.24	100.00	78.79	0.20	0.19
some	68	12293	36	5452	0.17	50.41	100.00	78.25	0.18	0.17
well	69	12546	36	5403	0.18	50.59	100.00	77.55	0.17	0.16
good	70	12861	36	5364	0.18	50.77	100.00	76.99	0.17	0.17
know	71	13284	36	5268	0.19	50.95	100.00	75.61	0.18	0.19
mr	72	29078	36	5249	0.41	51.36	100.00	75.34	0.37	0.30
before	73	10569	36	5238	0.15	51.51	100.00	75.18	0.15	0.15
time	74	10743	36	5235	0.15	51.66	100.00	75.14	0.16	0.16
own	75	11541	36	5191	0.16	51.82	100.00	74.51	0.15	0.15
never	76	11344	36	5122	0.16	51.98	100.00	73.52	0.16	0.16
your	77	24516	36	5095	0.34	52.32	100.00	73.13	0.28	0.24
did	78	11638	36	5088	0.16	52.48	100.00	73.03	0.17	0.16
say	79	11370	36	5084	0.16	52.64	100.00	72.97	0.15	0.13
must	80	11808	36	5069	0.17	52.81	100.00	72.76	0.15	0.15
man	81	13599	36	5062	0.19	53.00	100.00	72.66	0.19	0.19
made	82	9671	36	5045	0.14	53.13	100.00	72.41	0.13	0.13
other	83	9906	36	5038	0.14	53.27	100.00	72.31	0.14	0.14
only	84	9657	36	5035	0.14	53.41	100.00	72.27	0.14	0.13
upon	85	14349	36	5028	0.20	53.61	100.00	72.17	0.20	0.17
about	86	11439	36	4961	0.16	53.77	100.00	71.21	0.17	0.15
how	87	11409	36	4954	0.16	53.93	100.00	71.11	0.15	0.15
think	88	11609	36	4948	0.16	54.09	100.00	71.02	0.15	0.14
see	89	10867	36	4934	0.15	54.24	100.00	70.82	0.14	0.14
we	90	16139	36	4922	0.23	54.47	100.00	70.65	0.21	0.19
too	91	9451	36	4881	0.13	54.60	100.00	70.06	0.13	0.12
their	92	13495	36	4796	0.19	54.79	100.00	68.84	0.19	0.19
after	93	8446	36	4662	0.12	54.91	100.00	66.92	0.12	0.12
can	94	10751	36	4646	0.15	55.06	100.00	66.69	0.14	0.13
am	95	12636	36	4621	0.18	55.24	100.00	66.33	0.15	0.15
like	96	9713	36	4594	0.14	55.37	100.00	65.94	0.14	0.13
thought	97	8694	36	4583	0.12	55.50	100.00	65.78	0.12	0.12
might	98	8934	36	4536	0.13	55.62	100.00	65.11	0.13	0.11
make	99	8401	36	4520	0.12	55.74	100.00	64.88	0.11	0.10
may	100	10390	36	4514	0.15	55.88	100.00	64.79	0.12	0.10
has	101	12607	36	4497	0.18	56.06	100.00	64.55	0.15	0.13
great	102	8517	36	4445	0.12	56.18	100.00	63.80	0.12	0.11
come	103	9088	36	4439	0.13	56.31	100.00	63.71	0.12	0.12
over	104	7370	36	4210	0.10	56.41	100.00	60.43	0.11	0.10
himself	105	8760	36	4187	0.12	56.53	100.00	60.10	0.13	0.11
down	106	7952	36	4186	0.11	56.64	100.00	60.08	0.12	0.11
being	107	7329	36	4176	0.10	56.75	100.00	59.94	0.11	0.10
two	108	7345	36	4175	0.10	56.85	100.00	59.93	0.11	0.11
way	109	7082	36	4151	0.10	56.95	100.00	59.58	0.10	0.09
though	110	7134	36	4142	0.10	57.05	100.00	59.45	0.10	0.09
first	111	6785	36	4140	0.10	57.14	100.00	59.42	0.10	0.10
go	112	8238	36	4041	0.12	57.26	100.00	58.00	0.11	0.11
yet	113	7139	36	4033	0.10	57.36	100.00	57.89	0.09	0.08
ever	114	6898	36	4017	0.10	57.46	100.00	57.66	0.09	0.09
take	115	6628	36	3990	0.09	57.55	100.00	57.27	0.09	0.09
nothing	116	6880	36	3986	0.10	57.64	100.00	57.21	0.10	0.09
again	117	7340	36	3976	0.10	57.75	100.00	57.07	0.11	0.11
shall	118	9126	36	3954	0.13	57.88	100.00	56.75	0.10	0.09
day	119	7056	36	3944	0.10	57.97	100.00	56.61	0.10	0.10
these	120	6659	36	3939	0.09	58.07	100.00	56.54	0.09	0.09
without	121	6535	36	3932	0.09	58.16	100.00	56.44	0.10	0.09
most	122	6708	36	3886	0.09	58.25	100.00	55.78	0.10	0.09
every	123	7159	36	3794	0.10	58.35	100.00	54.46	0.10	0.08
last	124	5915	36	3763	0.08	58.44	100.00	54.01	0.09	0.09
give	125	6159	36	3708	0.09	58.52	100.00	53.22	0.08	0.07
came	126	6133	36	3669	0.09	58.61	100.00	52.66	0.09	0.09
here	127	6503	36	3630	0.09	58.70	100.00	52.10	0.09	0.08
house	128	7033	36	3618	0.10	58.80	100.00	51.93	0.10	0.10
where	129	6291	36	3602	0.09	58.89	100.00	51.70	0.09	0.10
long	130	5614	36	3567	0.08	58.96	100.00	51.20	0.08	0.09
hand	131	6176	36	3538	0.09	59.05	100.00	50.78	0.09	0.09
mind	132	5561	36	3524	0.08	59.13	100.00	50.58	0.08	0.08
our	133	7492	36	3511	0.10	59.23	100.00	50.39	0.10	0.10
mrs	134	12762	34	3501	0.18	59.41	94.44	50.25	0.17	0.14
life	135	5752	36	3490	0.08	59.49	100.00	50.09	0.08	0.08
us	136	7111	36	3472	0.10	59.59	100.00	49.83	0.09	0.09
better	137	5381	36	3462	0.08	59.67	100.00	49.69	0.08	0.07
indeed	138	5752	36	3451	0.08	59.75	100.00	49.53	0.07	0.06

```
let                    139      6166      36      3449      0.09   59.84   100.00    49.50     0.08    0.07
those                  140      5552      36      3437      0.08   59.91   100.00    49.33     0.07    0.07
myself                 141      6520      36      3426      0.09   60.00   100.00    49.17     0.08    0.08
always                 142      5622      36      3408      0.08   60.08   100.00    48.92     0.07    0.07
once                   143      4959      36      3403      0.07   60.15   100.00    48.84     0.07    0.07
away                   144      5585      36      3390      0.08   60.23   100.00    48.66     0.08    0.08
```

```
Parameter settings :
atomize   1
casefold  1
dateline  Fri Nov 13 16:41:42 2015
doclist   <class 'list'> of 36 items.
docpath   c:\vocsoft\samples\britfict\novs\
docpaths  c:\vocsoft\samples\britfict\novs\
docs      36
dumpname  C:\vocsoft\op\ew_vocs.dat
filetail  .txt
id        C:\vocsoft\p3\dox2vox.py
jobname   ew
minfreq   3
outpath   C:\vocsoft\op\
pathlist  ['c:\\vocsoft\\samples\\britfict\\novs\\']
progname  C:\vocsoft\p3\dox2vox.py
progpath  C:\vocsoft\p3\
punxtab   <class 'dict'> of 25 items.
snipsize  1024
sortcol   sniprate
topvocs   144
totsnips  6967
tottoks   7140077
vocdump   ew_vocs.dat
vocfile   ew_vocs.txt
vocsize   60503
voutname  C:\vocsoft\op\ew_vocs.txt
whereat   C:\vocsoft\p3\
wordonly  1
zonk      0
```

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.

ordering criterion	**meaning**
corprate	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'.
docrate	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.
sniprate	This is the percentage of 'snippets' (of size given by 'snipsize') in which the wordform is found.

textmean	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.
textmid	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.

```
wordform      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      146385 36     6967   2.05   15.21  100.00 100.00 2.10   2.16
in     6      113186 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 1.30   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     84883  36     6963   1.19   21.22  100.00 99.94  1.24   1.17
....
life   135    5752   36     3490   0.08   59.49  100.00 50.09  0.08   0.08
us     136    7111   36     3472   0.10   59.59  100.00 49.83  0.09   0.09
better 137    5381   36     3462   0.08   59.67  100.00 49.69  0.08   0.07
indeed 138    5752   36     3451   0.08   59.75  100.00 49.53  0.07   0.06
let    139    6166   36     3449   0.09   59.84  100.00 49.50  0.08   0.07
those  140    5552   36     3437   0.08   59.91  100.00 49.33  0.07   0.07
myself 141    6520   36     3426   0.09   60.00  100.00 49.17  0.08   0.08
always 142    5622   36     3408   0.08   60.08  100.00 48.92  0.07   0.07
once   143    4959   36     3403   0.07   60.15  100.00 48.84  0.07   0.07
away   144    5585   36     3390   0.08   60.23  100.00 48.66  0.08   0.08
```

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 display conveniently. (The full version is on the op subfolder.)

```
prepath       textname      filenum       totchars      tottoks       totvocs       bw
       diversim      haprate       herdanc       hr     v2overv       sniphaps
       shsd   sniptr        stsd   the_   and_   to_    of_    a_     in_    that
       with   as_    it_    for_   but_   not_   be_    at_    was_   have   had_   by_
       on_    all_   so_    this   i_     his_   he_    from   is_    which  no_    if_
       would  when   one_   what   an_    him_   you_   or_    been   my_    her_
       were   very   there  more   me_    said   who_   could  than   now_   she_
       out_   they   do_    any_   will   up_    them   are_   should into   much
       then   such   little some   well   good   know   mr_    before time   own_
       never  your   did_   say_   must   man_   made   other  only   upon   about
       how_   think  see_   we_    too_   their  after  can_   am_    like   thought
       might  make   may_   has_   great  come   over   himself       down   being
       two_   way_   though first  go_    yet_   ever   take   nothing       again
       shall  day_   these  without       most   every  last   give   came   here
       house  where  long   hand   mind   our_   mrs_   life   us_    better indeed
       let_   those  myself always once   away
c:\vocsoft\samples\ew\taletext    EW_AfterHolbein.txt 1      48995  8764   2086
       12.1199938    0.9883045     0.5982742     0.8418881     2259.0456372  0.1447747
       31.1401367    1.7268774     44.7265625    1.7312603     5.3400274     3.4345048
       2.7727065     2.247832      1.8370607     1.4376997     1.1980831     0.9128252
       0.8443633     1.1068005     0.4906435     0.5591054     0.5020539     0.2966682
       0.8557736     1.4947513     0.2510269     1.1866728     0.2396166     0.798722
       0.3537198     0.3080785     0.1255135     0.6617983     1.1410315     1.7800091
       0.2282063     0.1939754     0.2852579     0.3537198     0.2510269     0.1711547
       0.2510269     0.3765404     0.2510269     0.2966682     0.5248745     0.5476951
       0.3423094     0.3651301     0.1939754     1.5175719     0.3194888     0.1369238
       0.2053857     0.2053857     0.2510269     0.2738476     0.3194888     0.182565
       0.1369238     0.216796      1.1182109     0.4564126     0.2510269     0.182565
       0.0798722     0.0114103     0.3423094     0.1483341     0.0342309     0.0456413
       0.3194888     0.1026928     0.216796      0.0456413     0.2510269     0.0570516
       0.1939754     0.2053857     0.0912825     0.3765404     0.1369238     0.1026928
       0.0456413     0.1026928     0.0456413     0.1483341     0.1369238     0.0684619
       0.0912825     0.0684619     0.1939754     0.216796      0.0    0.2510269
       0.1369238     0.0570516     0.1141031     0.0570516     0.1939754     0.1369238
       0.216796      0.0342309     0.0    0.1939754     0.3080785     0.0684619
       0.0342309     0.0    0.0114103     0.0456413     0.0684619     0.1711547
       0.2510269     0.2738476     0.0798722     0.1939754     0.1026928     0.0570516
       0.0684619     0.1141031     0.0456413     0.0912825     0.0342309     0.0912825
       0.2053857     0.0684619     0.0570516     0.1026928     0.0342309     0.0570516
       0.1141031     0.1369238     0.0342309     0.0456413     0.0570516     0.1255135
       0.2053857     0.1939754     0.1369238     0.1255135     0.0    0.798722
       0.0342309     0.0228206     0.0912825     0.0    0.0684619     0.0684619
       0.0114103     0.3308991     0.0684619     0.0456413
```

Information concerning the output variables is given in the table below.

Column name	Contents
prepath	This is the directory path of the file concerned, up to but excluding the filename itself.
textname	This is the name of the text file concerned, without its directory path prefix.
filenum	This is a serial number, giving the order in which the files were processed.
totchars	This is the total number of characters in the file.
tottoks	This is the total number of tokens (which might not always be words) in the file as computed by the program's tokenizer.
totvocs	This is the total number of distinct tokens found in the file, i.e. the vocabulary size.

bw	This is Brunet's W (Brunet, 1978), a measure of vocabulary richness computed as W = N ^ (V ^ (-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.)
diversim	This is Simpson's index of diversity (see Upton & Cook, 2006) $S = 1 - \sum(p_j \wedge 2)$ 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.
haprate	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.
herdanc	This is the bilogarithmic Type-Token ratio, also known as Herdan's C (Herdan, 1960), computed as C = ln(V) / ln(N) and it too, alas, varies systematically with text length.
hr	This is Honoré's R (Honoré, 1979), computed as (100 * ln(N)) / (1 - H/(V+0.5)) where H is the number of *Hapax Legomena* and N and V are as above. This index is relatively stable across text lengths (above about 1200 tokens) so can be used to compare vocabulary richness among texts of various sizes (Holmes, 1994; Tweedie & Baayen, 1998).
v2overv	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).
sniphaps	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.
shsd	This gives the standard deviation of the values used to compute sniphaps.
snipttr	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).
stsd	This gives the standard deviation of the values used in computing snipttr.
.... the remainder	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.

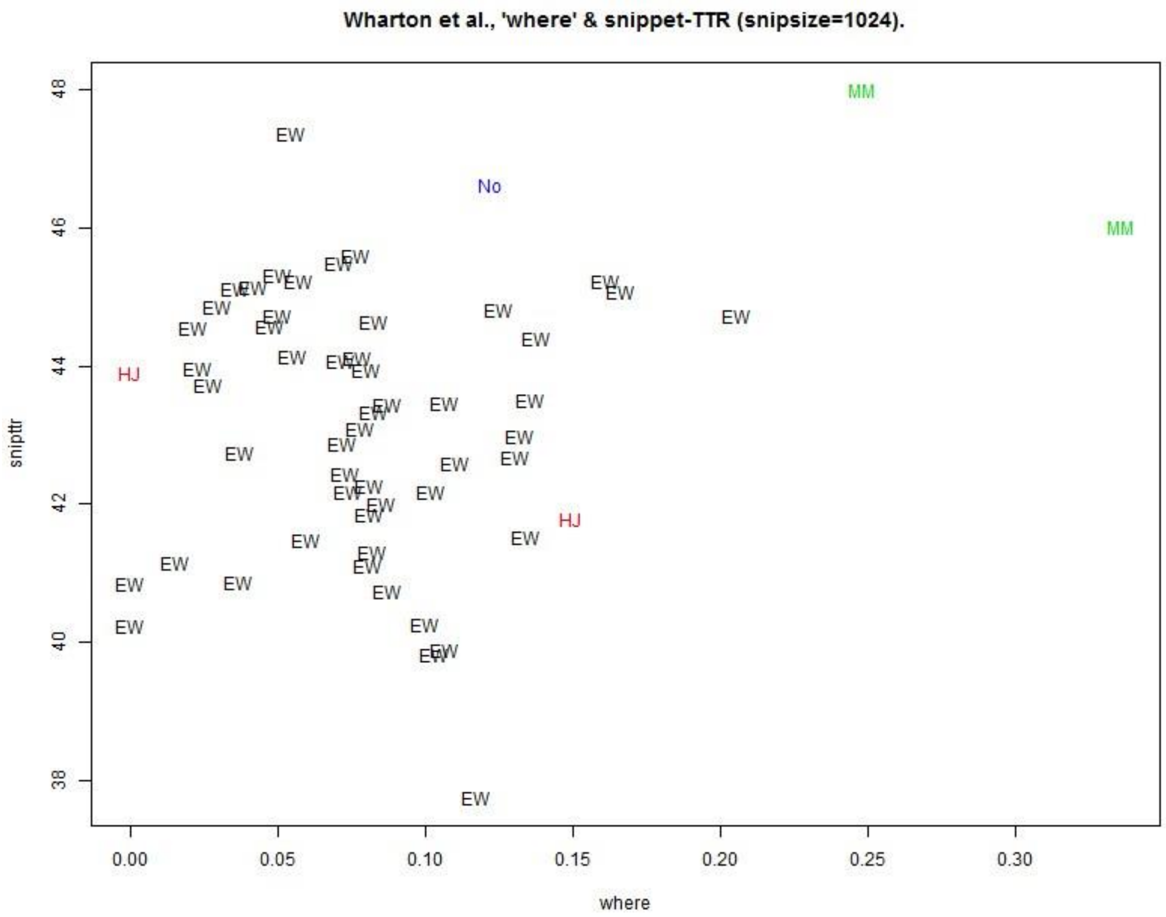
What can be done with this kind of output?

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")
```

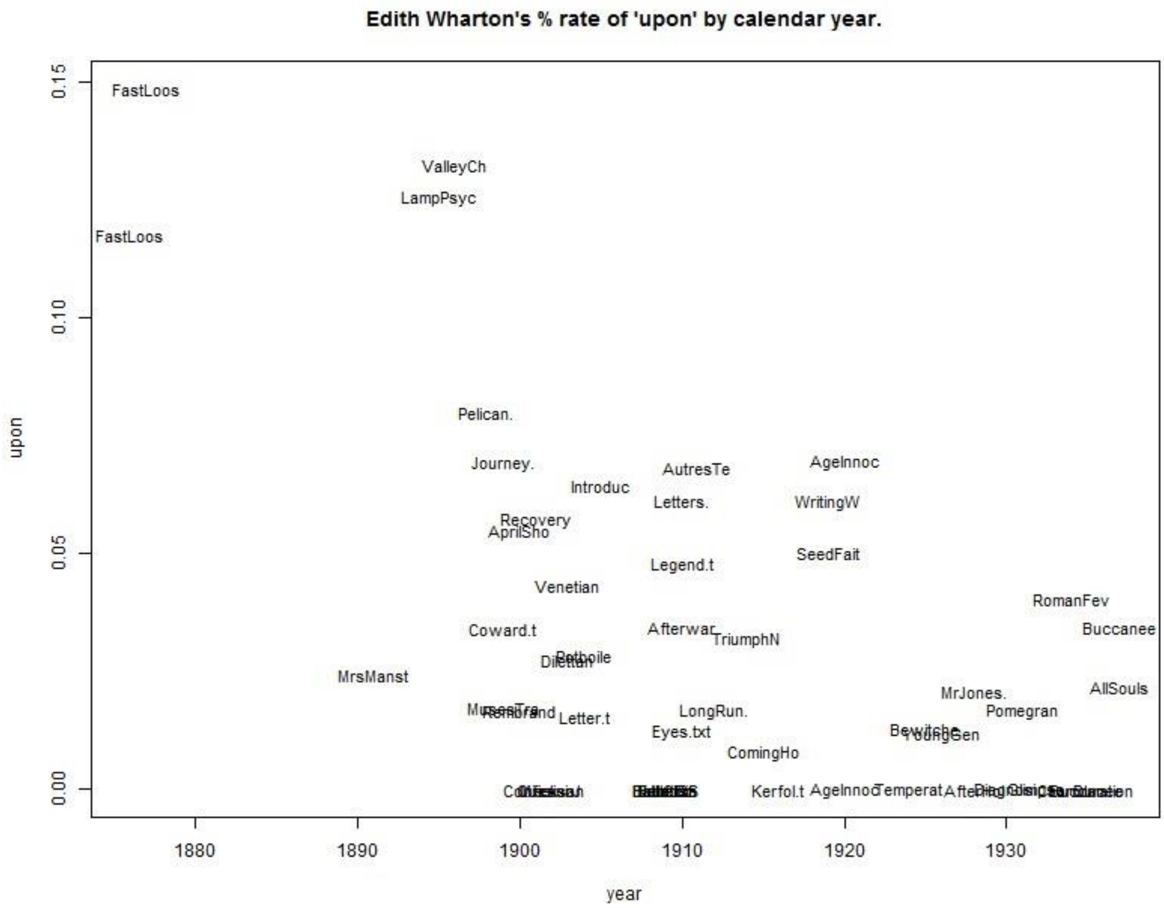
would read the data file created above into a data-frame called ew with 57 rows and 160 columns. Incidentally, the first 52 rows of this file refer to writings by Edith Wharton and the last five by comparison authors. These five include two chapters by Henry James (a mentor of Wharton) and two by Marion 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.

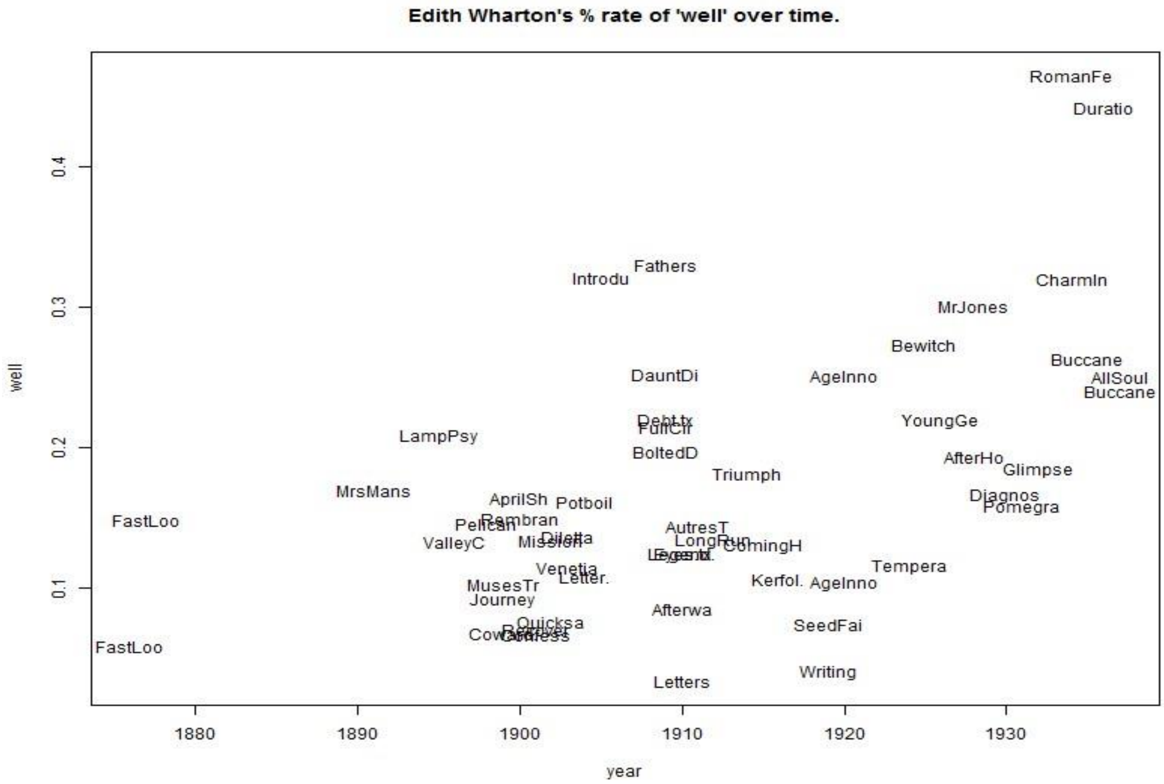
A scatter plot produced in R of these 57 texts, where the axes are the values of snipttr and the percentage occurrence rate of the word 'where', is shown below.



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.



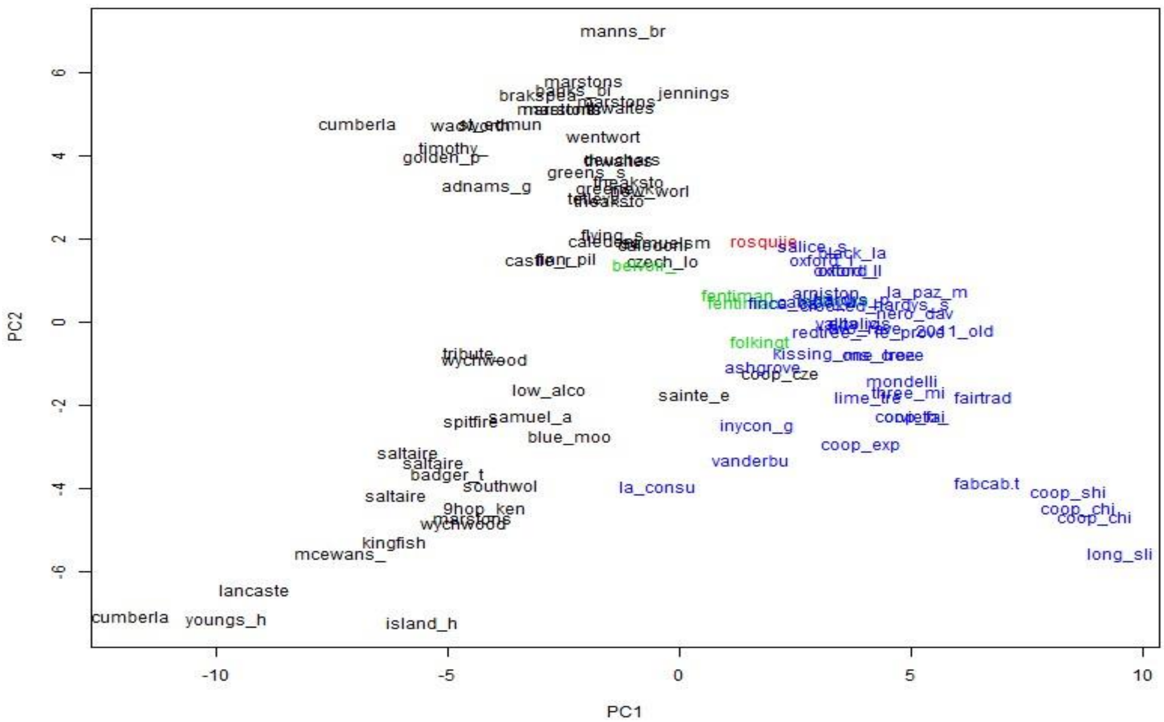


Another example is illustrated below. This arises from the bottlabs corpus, a collection of short texts taken from the back-labels of beverage bottles. In this case, after creating a vocabulary of 144 items with dox2vox.py and then creating a data grid with vox2dat.py, the resultant data grid was read into R and subjected to a Principal Components Analysis (PCA) using the prcomp() built-in function.

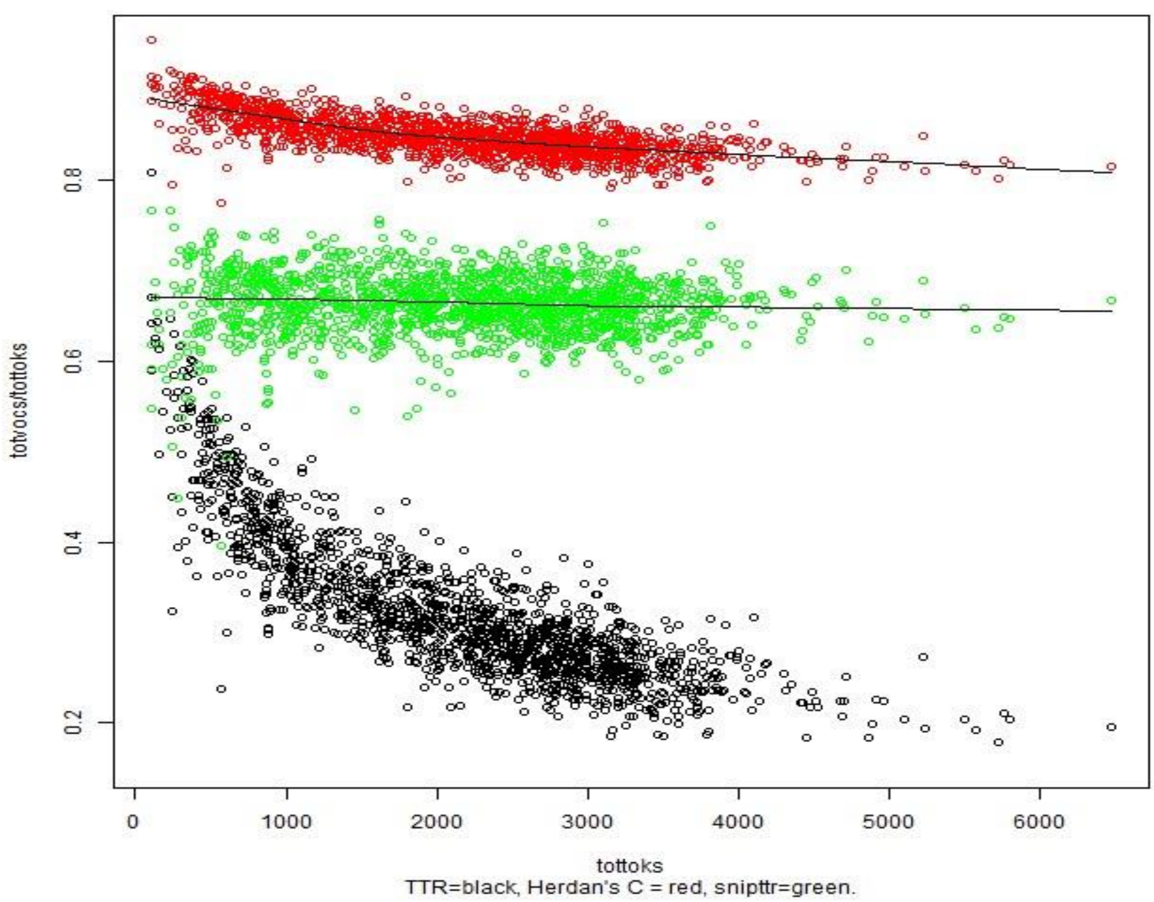
The graph shows each of the file names (truncated to 8 characters) positioned in the space of the first 2 principal components, which account for 22% or the overall variance. Beers are in black, wines are coloured blue, soft drinks are in green and the sole cider is coloured red. The main categories, 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 sniptt r (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 sniptt r. (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, sniptt r is much less strongly associated with number of tokens, though even here a slight downward trend is discernible.

botlabs, topvocs=144, first 2 principal components.



TED.com vocabulary richness indices by text length.



Acknowledgements

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

Thank you for reading this far. :-)

References

Brunet, E. (1978). *Vocabulaire de Jean Girandoux: Structure et Évolution*. Paris: Slatkine.

Burrows, J.F. (1987). Word-patterns and story-shapes: the statistical analysis of narrative style, *Literary and Linguistic Computing*, 2, 61-70.

Burrows, J.F. (1992). Not unless you ask nicely*:* The interpretive nexus between analysis and information*. Literary and Linguistic Computing*, 7, 91–109.

Burrows J.F & Craig D.H. (2001). Lucy Hutchinson and the authorship of two seventeenth-century poems: a computational approach. *The Seventeenth Century,* 16, 259-282.

Forsyth, R.S., Holmes, D.I. & Tse, E.K. (1999). Cicero, Sigonio and Burrows: investigating the authenticity of the Consolatio. *Literary & Linguistic Computing,* 14(3). 375-400.

Herdan, G. (1960). *Type-Token Mathematics*. Mouton: The Hague.

Holmes, D.I. (1994). Authorship attribution. *Computers and the Humanities*, 28(2), 87-106.

Holmes, D.I. & Forsyth, R.S. (1995). The Federalist revisited: new directions in authorship attribution. *Literary & Linguistic Computing*, 10(2), 111-127.

Honoré, A.M. (1979). Some simple measures of richness of vocabulary. *ALLC Bulletin*, 7(2), 172-177.

Mosteller, F. & Wallace, D.L. (1984). *Applied Bayesian and Classical Inference: The Case of the Federalist Papers*. New York: Springer-Verlag.

R Core Team (2013). R: A language and environment for statistical computing. *R Foundation for statistical Computing*, Vienna, Austria.
http://www.R-project.org/.

Sichel, H.S. (1975). On a distribution law for word frequencies. *Journal of the American Statistical Association*, 70, 542-547.

Fiona J. Tweedie and R. Harald Baayen (1998). How variable may a constant be? Measures of lexical richness in perspective. *Computers and the Humanities*, 32:323–352.

Upton, G. & Cook, I. (2006). *Oxford Dictionary of Statistics*, second ed. Oxford: Oxford University Press.

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 parameters, 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.

```
##  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
filetail  .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
progpath  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 'corprate'. (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.