

TOCCATA : Text-Oriented Computational Classifier Applicable To Authorship

(User Notes by Richard Forsyth, June 2015)

Toccata is a system for testing text-classification techniques, written in Python3. Essentially the main program is a test harness into which a variety of text-classification algorithms can be inserted for evaluation on unproblematic cases and, if required, applied to disputed cases.

Why I Wrote this Software

In the 20 years or so since I became interested in computational authorship attribution, I have implemented several algorithms to perform text categorization (k-nearest-neighbour, linear classifier, naive Bayes, tree-induction, among others) in a variety of programming languages, including Basic, C, Python2, Python3, R and Snobol4/Spitbol. This left me with a motley collection of programs, most of which I can no longer execute due to lack of support software, all of which have irritatingly different conventions about input formats and operational parameters.

I realized that what I wanted was a generic framework into which I could plug alternative classification techniques. That would allow me to evaluate the success of a possibly novel technique on a common corpus of documents with undisputed class membership; and, if it appeared promising, to apply it to unseen or genuinely problematic cases.

Toccata is the result. (The name stands for: Text-Oriented Computational Classifier Applicable To Authorship.) I am making it available to all & sundry as freeware in the GNU sense of 'free' in the hope that it will be useful to others, and possibly even lead to paid consultancy -- since software can always be improved or extended and there might even be people prepared to pay me to do the extending &/or improving. :-)

As mentioned above, my main motive in writing toccata was to experiment with authorship-attribution methods, but it can do many other kinds of document categorization as well, e.g. classifying by topic or by genre.

The basic concept is that you write a classifier as a Python3 library and run it through the toccata main program which tries it out on a test corpus or corpora and calculates a number of evaluation measures, as well as classifying a holdout sample if present. Actually, you don't have to write your own classifier, since four different (simple but quite effective) libraries are supplied so that those who don't fancy writing Python code can still use the system for document classification. If you are happy to write Python code, these serve as models which you can adapt for your own purposes.

A Sketch of the System's Operation

Phase	Brief Outline
00.	Collect text data. Can't say much about this except that it could take lots of work, and that each document should be in a separate (utf8) file and should belong to a specific category. Four example corpora are provided to get you started. (Incidentally, data gathering & checking is the really crucial part: doing insufficient data validation is a trap into which almost everyone has fallen at some time, including me.)
0a.	Download Python (version 3 not 2), if you don't already have it, from www.python.org . This is normally quite painless.
0b.	Unpack the toccata.zip file -- into a top-level directory called toccata unless you want to do lots of extra editing.
1a.	Create a "metafile" for the training-set of documents. A program called metaget.py is provided to help make this process (fairly) easy. More on metafiles below.

1b.	Optional, but very likely: also create a metafile for a holdout sample of texts, including some of uncertain category membership.
2.	Either write your own bespoke text-classifier as a Python3 module, or (more likely at first) decide which of the provided library modules, <code>docalib_deltoid</code> , <code>docalib_keytoks</code> , <code>docalib_maws</code> or <code>docalib_topvocs</code> , to use. More details below.
3.	Prepare a parameter file. This is a file that can be edited, e.g. in Notepad or Notepad++, which specifies various settings. Examples of parameter files will be shown below.
4.	Run <code>toccata7.py</code> . (The digit is a version number, so 7 may change to 8 or more as time goes by.) This performs three main functions, in sequence: (a) <code>testmode</code> : leave-n-out random resampling test of the classifier on the training corpus to provide statistics by which the classifier can be evaluated; (b) <code>holdout</code> : application of the classifier to an unseen holdout sample of texts, if a test metafile is given; (c) <code>posthoc</code> : re-application to the holdout sample of texts (if one is given) using the results from the <code>testmode</code> phase to estimate empirical probabilities. More details below. Note that steps (b) & (c) are optional. Note also that step (c) is frankly experimental thus needs to be treated with caution.
5.	Peruse results with care, perhaps exporting the <code>"_dump"</code> file into R or another statistical package for further processing.

Phase 00 : Corpus Format

Toccata is a document-oriented system. Thus a training corpus consists of a number of text files (in UTF8 encoding). Each file is treated as an individual document, belonging to a particular category.

In the samples folder you will find four subfolders (`ajps`, `cics`, `feds`, and `mags`). These contain datasets that enable you to start using the system, prior to collecting &/or reformatting your own corpora.

The first, `ajps`, contains ninety poems by 2 eminent 19th-century Hungarian poets, Arany József & Petőfi Sándor. Arany was godfather to Petőfi's child, so we might expect their writing styles to be relatively similar. Also, these poems are short compared to the lengths of documents that are typically used in text classification, so represent a challenging problem.

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. We will use this dataset as our main example to demonstrate how toccata works.

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 `mags` subfolder contains data for a content-discrimination problem. It contains 144 texts from 2

different learned journals, namely *Literary & Linguistic Computing* and *Machine Learning*. Each text is an excerpt consisting of the Abstract plus initial paragraph of an article in one of those journals, written during the period 1987-1995. The classification task is to decide the journal in which the text was published. Hence this is not an authorship problem, rather a problem of content discrimination. Again the texts are relatively short compared to other examples in this field.

Phase 0 : 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 toccata.zip file. After unpacking it, preferably into a folder called "toccata", you should find the following subfolders.

```
op
p3
parapath
samples
```

The programs are in p3. Sample test corpora 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.

Phase 1 : Anything You Can Do, I Can Do Meta (-:-)

Sorry, couldn't resist that.

Below is a complete listing of a training metafile for the cics dataset. It has three columns. A metafile could have more columns than three, but not less. The top line is a header, giving the column names. The first column must be called prepath. It indicates the directory/folder where a particular file resides. The second must be called filename and will contain the file names of each particular text. The other column will contain class labels. It can be called anything, though doctype is the default. (See details of parameter files, in Phase 3, for alternative ways of indicating the class of a text.) Columns are separated by the horizontal tab character. (Code point 9 in ASCII and Unicode/utf8.) Each line after the header refers to a separate document.

```
prepath      filename      doctype
c:\toccata\samples\cics\Tullies\ Cicero_Amicitia.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_ArchiaPoeta.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Atticus1.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Brutus1.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Brutus2.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Cat2.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_CatoSenectute.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_DeFinibus.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_DeImperio.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_DeInventione2_latlib.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_DeLegibus.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_DePartitione_latlib.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_InPisonem_latlib.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_InVerremII2_latlib.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_NaturaDeorum2.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Officiis1.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Orator.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_Philippics2.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_ProCaecina_latlib.txt cics
c:\toccata\samples\cics\Tullies\ Cicero_ProCluentio.txt cics
```

c:\toccata\samples\cics\Tullies\	Cicero_ProFlacco_latlib.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_ProMarcello.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_ProMilone_latlib.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_ProQuinctio_latlib.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_ProSestio_latlib.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_ProSexto.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_ProSulla.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_Rep2.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_Somnium.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_Tusculan1.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_Tusculan2.txt	cics
c:\toccata\samples\cics\neolats\	Muretus_PaulFox.txt	muretus
c:\toccata\samples\cics\neolats\	Muretus_Phil.txt	muretus
c:\toccata\samples\cics\neolats\	Muretus_Pius.txt	muretus
c:\toccata\samples\cics\neolats\	Muretus_Rege.txt	muretus
c:\toccata\samples\cics\neolats\	Muretus_Util.txt	muretus
c:\toccata\samples\cics\neolats\	Sigonio_Elo1.txt	sigonio
c:\toccata\samples\cics\neolats\	Sigonio_Elo2.txt	sigonio
c:\toccata\samples\cics\neolats\	Sigonio_HistIt4a.txt	sigonio
c:\toccata\samples\cics\neolats\	Sigonio_HistIt4b.txt	sigonio
c:\toccata\samples\cics\neolats\	Sigonio_LatLing.txt	sigonio
c:\toccata\samples\cics\neolats\	Sigonio_LaudHist.txt	sigonio

This metafile describes a training corpus with 3 categories: 31 texts by Cicero, 5 texts by Muretus and 6 texts by Sigonio. Many of these 42 texts are extracts rather than full works. Note that no disputed texts are included in the training corpus. Note also that only 5 or 6 training examples is much fewer than ideal, so it is optimistic to expect high accuracy in this case; however, in real problems we are often forced to compromise. (The program cannot run with fewer than 2 instances of each training category.)

There follows a complete listing of holdout3.txt, a testing metafile for this example. This does include disputed texts.

prepath	filename	doctype
c:\toccata\samples\cics\claslats\	Seneca_Brevit.txt	claslats
c:\toccata\samples\cics\claslats\	Seneca_Cons.txt	claslats
c:\toccata\samples\cics\claslats\	Seneca_Ira1.txt	claslats
c:\toccata\samples\cics\claslats\	Seneca_Otio.txt	claslats
c:\toccata\samples\cics\claslats\	Seneca_Prov.txt	claslats
c:\toccata\samples\cics\neolats\	Abelard_HistCalamitatum_latlib.txt	neolats
c:\toccata\samples\cics\neolats\	Heloise_Epistola_latlib.txt	neolats
c:\toccata\samples\cics\neolats\	Lauredan_FranVen.txt	neolats
c:\toccata\samples\cics\neolats\	Lauredan_Mant.txt	neolats
c:\toccata\samples\cics\neolats\	Muretus_Ingress.txt	muretus
c:\toccata\samples\cics\neolats\	Muretus_Laud.txt	muretus
c:\toccata\samples\cics\neolats\	Sigonio_Dialogo.txt	sigonio
c:\toccata\samples\cics\Tullies\	Cicero_Philippics7.txt	cics
c:\toccata\samples\cics\Tullies\	Cicero_Tusculan4.txt	cics
c:\toccata\samples\cics\holdout\	ConsolA.txt	cons
c:\toccata\samples\cics\holdout\	ConsolB.txt	cons
c:\toccata\samples\cics\holdout\	EpistulaOct.txt	fake
c:\toccata\samples\cics\holdout\	RhetHerr.txt	fake

The last four entries refer to the first and second halves of the 1583 *Consolatio*, as well as 2 classical works, supposedly written by Cicero, which are nowadays taken to be spurious. Note that none of these have a category label seen in the training metafile. There are also several classical and neolatin "distractors" as well as one unseen text by Sigonio, 2 by Muretus and 2 by Cicero. As far as this holdout sample is concerned, the classifier cannot get more than five of its responses correct. However, it is interesting to observe how it handles the distractors.

The format of metafiles is intended to be suitable for manipulation in a spreadsheet package such as

Excel or OpenOffice/Calc as a tab-delimited worksheet. The idea behind this is to make it possible to select a variety of subsets of a larger corpus as training or test texts in different runs of the system, without moving files around &/or deleting them.

To make an initial metafile, it is convenient to use the `metaget.py` program, which is included with the distribution. The output of this program can then be edited in a text-editor, such as Notepad++, or a spreadsheet until it specifies exactly the desired set of files. Notepad++, a versatile text-editor that I personally recommend, can be obtained from the website <http://notepad-plus-plus.org/> free of charge.

The `metaget.py` program can be run just by double-clicking on its name. It will then display a window with four elements:

Enter next category name:
Select file(s):
Enter output metafile name:
Exit & save metafile:

The idea is that you type a category label in the upper box (then press the Enter button) then choose files by picking the second option which will allow the customary ways of navigating the file system and selecting files or groups of files. This pair of actions can be repeated several times to include files from a number of different categories &/or different folders. Then you provide a destination file name and extension for the resulting metafile (again not forgetting to press the Enter button) and quit using the final option. If you do forget to name the output metafile, it will be called `metazero.txt` and placed on the directory from which the program was launched.

Note that entering the category or metafile name does require **clicking the Enter button** alongside the text-entry box to confirm your input; just hitting Carriage-Return won't do, as I have yet to master the intricacies of binding a keypress-response procedure to the Return key. (Still writing programs as if the 20th century hadn't gone out of fashion, I'm afraid. Nevertheless, I suspect most people will find `metaget.py` somewhat simpler to use than its precursor `minimet4.py`, though I doubt if it will eliminate cases where using a text-editor, such as Notepad++, will still be needed to put a nearly-correct metafile into its final form.)

The four test problems in the samples subfolder contain several metafiles that you can inspect as examples before making your own.

Phase 2 : Library Modules

Here we just consider the libraries provided with the system. For those dauntless spirits who enjoy writing modules in Python3, Appendix 4 gives much fuller details of what a library should provide for the `toccata7.py` program (essentially a class called `Docadat` which includes a number of required methods that create and employ a list called `modinfo` of models for each category) and what data structures the `toccata7.py` program makes readable to the methods in that class (essentially a list called `doclist`, containing details of each text, and an object called `paradat` which holds the main program's parameter values). Somehow or other, each module must be capable of computing a matching score between any text and a category model. This score should be higher, more positive, the more closely the text matches the model. (It does not need to be proportional to a probability.)

Realistically, however, there is no need for such efforts, certainly not to begin with, since four library modules exist already "off the shelf", to get you started:

docalib_deltoid, docalib_keytoks, docalib_maws.py and docalib_topvocs.py.

Module **docalib_deltoid.py** is an implementation of Burrows's delta (Burrows, 2002) which has become a standard technique in authorship attribution studies. In a nutshell, this method first finds the most frequent N word tokens in the corpus; then computes the standard deviations of the relative usage rates of these words across the various documents of the corpus. This allows it to consider the mean usage rates of these words in each category as a model of that class. To compare a single text with a class model, it computes the absolute z-scores of all these words and averages them, a z-score being computed by subtracting the usage rate of the word under consideration in the text from the mean rate in the class model and dividing this difference (ignoring sign) by the standard deviation of that word in the corpus as a whole. This process yields a mean absolute z-score, which is a dissimilarity measure. Because **toccata7.py** works with similarities, these mean dissimilarities (d_i) are converted to similarities as $1.0/d_i$. The number, N , of most-frequent words to employ can be set using the `paraline` parameter (see Appendix 2) but if this is absent the system sets N to be the square root of the vocabulary size (total different vocabulary items, not total running tokens), which is usually a reasonable choice.

Library module **docalib_keytoks.py** is the method that does best on my personal benchmark collection of authorship problems. It works by first finding the 1024 most common word tokens in the corpus, then keeping from these the most distinctive. Distinctiveness is scored by comparing each class in turn with the aggregate of the other classes, using the measure $p \cdot q$, where p is the proportion of in-class snippets in which the token is found and q is 1 minus the proportion of other-class snippets in which the token is found, i.e. the proportion in which it isn't found. A snippet is a sonnet-sized sequence of 115 words by default. Having ranked these tokens by this score, the N items from both ends of the ranking list are selected, where N can be given by the user but by default is the square root of the total vocabulary size. The resultant set of keywords is the union of those picked for each class. For classification, the frequencies of these selected keywords in the text being classified are correlated (with Spearman's rho by default) with the relative frequencies of these terms in each class. The class assigned is that with highest correlation. The method tends to employ quite large numbers of words. Perhaps surprisingly, even when correlating several hundred words, many on tied ranks with low frequencies such as 0, 1 or 2 in the text being classified, this method gives quite accurate results.

The library module **docalib_maws.py** contains data and methods inspired by what Mosteller & Wallace (hence MAWS, Mosteller And Wallace System) in their classic work (1964/1984) on the disputed Federalist papers call their "robust Bayesian analysis". I have slightly revised the software which I wrote originally in my 1995 thesis (Forsyth, 1995) to automate this approach. Essentially this is a naive Bayesian classifier using frequent word tokens. It takes 2 parameters, `toks2get` and `multivox` (default values 144 and 1.618034 respectively) and computes the rounded value of `toks2get` multiplied by `multivox`. It then takes the resulting number of the most common words in the corpus, according to document frequency, and reduces them to `toks2get` again by discarding the least discriminatory items. Then Bayes factors are computed from the training data for each remaining token according to how often the relative frequency of that token exceeds the median frequency of that token in the whole corpus.

The library module **docalib_topvocs.py** implements a classifier inspired by the approach of Burrows (1992). It uses the most frequent word-tokens in the training corpus as features. The number of most frequent words used in the feature list defaults to the (rounded) square root of the vocabulary size of the entire corpus, i.e. of the number of distinct word types. To compute a matching score between a text and a model, the correlation between the frequencies of the words in the common-word feature list of both the text and the model is computed. The user can select either Pearson's r

or Spearman's rank correlation coefficient. Rank correlation is the default. (Using correlations is what differentiates this technique from most classifiers in the Mosteller/Wallace/Burrows tradition.)

Phase 3 : Preparing a Parameter File

Below is a listing of parameter file cicsvocs.txt which comes with the toccata distribution on the parapath folder.

```
comment  testing with Ciceronian corpus metafile :
jobname  cicsvocs
trainmet  c:\toccata\samples\cics\metadat\cictrain.txt
testmeta  c:\toccata\samples\cics\metadat\holdout3.txt
wordonly  1
libname  docalib_topvocs
```

A parameter file is just a plain text file with one item per line. Each line should begin with the parameter name, then 1 or more blank spaces, then the parameter value. The following table interprets the above parameter file, line by line.

Parameter	Default value	Function
comment	[None]	This (or in fact any unrecognized parameter name, e.g. "##") can be used to insert reminders about what the file is meant to do.
jobname	toccata	This gives the job a name. Any text string can be the value. It isn't necessary but it is useful as the jobname will be used as a prefix to the program's output files, so it can be seen that they form a group.
trainmet	[None]	This should be the full file specification of a metafile that indicates the text files that belong to the training corpus.
testmeta	[None]	This should be the full file specification of a metafile that indicates the holdout sample. It is optional: if omitted, the program only does the leave-n-out testing step.
wordonly	0	This should be integer 0 or 1. If it is 1, the tokenizer will ignore tokens unless they begin with an alphanumeric character. If it is zero, all tokens will be considered, even sequences of punctuation symbols and so on. Since punctuation of works authored by Cicero was definitely not Cicero's, it is set to 1 here.
libname	docalib_topvocs	This should be the name either of one of the supplied classifier libraries (e.g. docalib_deltoid, docalib_keytoks, docalib_maws.py or docalib_topvocs.py) or a user-written library. Don't include the .py suffix, as this is appended automatically by Python.

Phase 4 : Running the Main Program!

When you execute toccata7.py, for example with a command such as that below (shown in **bold**), you should see something like the following lines

```
C:\2015> python c:\toccata\p3\toccata7.py
C:\toccata\p3\toccata7.py 7.0 Thu Jun 25 14:29:34 2015
command-line args. = 1
prepath : C:\toccata\p3
working folder: C:\2015
script usage: python C:\toccata\p3\toccata7.py <parafilename>
please give parameter file name :
```

on the screen. If your parameter file is in the same directory as the program or in toccata's parapath subfolder you won't have to give the full path specification, just its name (.txt extension presumed).

Using the parameter file, cicsvocs.txt, shown in the previous section on the cics dataset produces four output files. The most important will normally have a name composed of the jobname, then "_list", with .txt extension. A listing of the output file cicsvocs_list.txt is shown below. (The other three output files include a dump of data that could be exported into R or a comparable statistical package. More on them later.)

```
dateline Thu Jun 25 14:15:19 2015
id C:\toccata\parapath\cicsvocs.txt
libname docolib_topvocs
progrname C:\toccata\p3\toccata7.py
targvar doctype
testmeta c:\toccata\samples\cics\metadat\holdout3.txt
trainmet c:\toccata\samples\cics\metadat\cictrain.txt
```

==== Subsampling trial :

rank	weight	filename	pred:true	predval	meanval
1	0.2077	Cicero_Philippics2.txt	cics + cics	0.7923	0.5847
2	0.2044	Cicero_Brutus1.txt	cics + cics	0.7230	0.5186
3	0.1866	Cicero_Brutus2.txt	cics + cics	0.7650	0.5784
4	0.1544	Cicero_Atticus1.txt	cics + cics	0.7841	0.6297
5	0.1464	Cicero_ProQuinctio_lat	cics + cics	0.7062	0.5598
6	0.1295	Cicero_InPisonem_latli	cics + cics	0.6734	0.5439
7	0.1257	Cicero_ProFlacco_latli	cics + cics	0.7126	0.5869
8	0.1254	Cicero_CatoSenectute.t	cics + cics	0.7463	0.6209
9	0.1151	Cicero_ProCaecina_latl	cics + cics	0.7446	0.6295
10	0.1147	Cicero_ProMarcello.txt	cics + cics	0.6793	0.5646
11	0.1122	Cicero_ProSulla.txt	cics + cics	0.6660	0.5538
12	0.1068	Sigonio_LaudHist.txt	sigonio + sigonio	0.7204	0.6136
13	0.1041	Sigonio_LatLing.txt	sigonio + sigonio	0.7550	0.6509
14	0.1027	Cicero_DeLegibus.txt	cics + cics	0.7782	0.6754
15	0.1026	Cicero_ProMilone_latli	cics + cics	0.5992	0.4966
16	0.0967	Cicero_Tusculan2.txt	cics + cics	0.7411	0.6445
17	0.0938	Cicero_Amicitia.txt	cics + cics	0.7643	0.6705
18	0.0877	Cicero_DeInventione2_1	sigonio - cics	0.5994	0.5117
19	0.0874	Sigonio_Elo1.txt	sigonio + sigonio	0.7060	0.6186
20	0.0770	Cicero_InVerremII2_lat	cics + cics	0.7444	0.6674
21	0.0638	Muretus_Pius.txt	muretus + muretus	0.6925	0.6286
22	0.0608	Muretus_PaulFox.txt	muretus + muretus	0.6949	0.6341
23	0.0600	Sigonio_Elo2.txt	sigonio + sigonio	0.6561	0.5962
24	0.0597	Muretus_Phil.txt	sigonio - muretus	0.6788	0.6192
25	0.0587	Cicero_Tusculan1.txt	cics + cics	0.7099	0.6512
26	0.0574	Cicero_ProSexto.txt	cics + cics	0.6524	0.5950
27	0.0568	Muretus_Rege.txt	muretus + muretus	0.6265	0.5697
28	0.0563	Cicero_DePartitione_la	cics + cics	0.5574	0.5010
29	0.0518	Cicero_DeFinibus.txt	cics + cics	0.6915	0.6397
30	0.0503	Cicero_Orator.txt	cics + cics	0.6474	0.5971
31	0.0389	Cicero_Officiis1.txt	sigonio - cics	0.6745	0.6356
32	0.0387	Cicero_ProSestio_latli	cics + cics	0.6139	0.5751
33	0.0387	Cicero_Somnium.txt	cics + cics	0.5967	0.5580
34	0.0385	Muretus_Util.txt	muretus + muretus	0.7062	0.6676
35	0.0369	Sigonio_HistIt4b.txt	sigonio + sigonio	0.6167	0.5797
36	0.0343	Cicero_ProCluentio.txt	cics + cics	0.6068	0.5725
37	0.0329	Sigonio_HistIt4a.txt	sigonio + sigonio	0.6154	0.5825
38	0.0189	Cicero_ArchiaPoeta.txt	sigonio - cics	0.6859	0.6671
39	0.0159	Cicero_Cat2.txt	sigonio - cics	0.5642	0.5483
40	0.0154	Cicero_Rep2.txt	sigonio - cics	0.5948	0.5795
41	0.0117	Cicero_DeImperio.txt	cics + cics	0.6430	0.6313
42	0.0111	Cicero_NaturaDeorum2.t	sigonio - cics	0.6543	0.6432

+++++-----

Confusion matrix :


```

Truecat =          cics muretus sigonio
Predcat : cics      159      0      2
Predcat : muretus    2      25      7
Predcat : sigonio    26      6     31

```

```

Kappa value = 0.6595
Precision (%) by category :
cics      98.7578
muretus    73.5294
sigonio    49.2063
Recall (%) by category :
cics      85.0267
muretus    80.6452
sigonio    77.5

```

```

cases = 258
cases with unseen category labels = 0
hits = 215
percent hits = 83.33
mean entropy = 1.4211
mean spherical score = 0.6441
point-biserial correlation of deltas = 0.7277

```

==== Holdout trial :

rank	weight	filename	pred:true	predval	meanval
1	0.0981	Cicero_Philippics7.txt	cics + cics	0.6350	0.5368
2	0.0972	EpistulaOct.txt	cics ? fake	0.5916	0.4945
3	0.0865	Muretus_Ingress.txt	muretus + muretus	0.6642	0.5777
4	0.0800	Lauredan_FranVen.txt	sigonio ? neolats	0.7536	0.6736
5	0.0605	Muretus_Laud.txt	muretus + muretus	0.7261	0.6656
6	0.0493	Lauredan_Mant.txt	sigonio ? neolats	0.6927	0.6433
7	0.0390	Sigonio_Dialogo.txt	sigonio + sigonio	0.6864	0.6473
8	0.0372	Cicero_Tusculan4.txt	cics + cics	0.6935	0.6563
9	0.0335	Seneca_Iral.txt	cics ? claslats	0.4870	0.4535
10	0.0304	ConsolA.txt	muretus ? cons	0.6425	0.6121
11	0.0281	Seneca_Prov.txt	muretus ? claslats	0.5401	0.5120
12	0.0279	Seneca_Otio.txt	sigonio ? claslats	0.5546	0.5268
13	0.0275	RhetHerr.txt	cics ? fake	0.4671	0.4395
14	0.0263	Seneca_Brevit.txt	cics ? claslats	0.6248	0.5985
15	0.0242	Seneca_Cons.txt	muretus ? claslats	0.6000	0.5759
16	0.0212	ConsolB.txt	muretus ? cons	0.6682	0.6469
17	0.0133	Abelard_HistCalamitatu	sigonio ? neolats	0.5482	0.5349
18	0.0121	Heloise_Epistola_latli	muretus ? neolats	0.5242	0.5121

+?+?+?+?+?+?+?+?+?+?

Confusion matrix :

Truecat =	cics	claslats	cons	fake	muretus	neolats	sigonio
Predcat : cics	2	2	0	2	0	0	0
Predcat : claslats	0	0	0	0	0	0	0
Predcat : cons	0	0	0	0	0	0	0
Predcat : fake	0	0	0	0	0	0	0
Predcat : muretus	0	2	2	0	2	1	0
Predcat : neolats	0	0	0	0	0	0	0
Predcat : sigonio	0	1	0	0	0	3	1

```

Kappa value = 1.0
Precision (%) by category :
cics      33.3333
muretus    28.5714
sigonio    20.0
Recall (%) by category :
cics      100.0
claslats    0.0
cons        0.0
fake        0.0
muretus    100.0

```

```

neolats      0.0
sigonio      100.0

cases = 18
cases with unseen category labels = 13
cases with known category labels = 5
[results below till * only apply to these 5 cases]
hits = 5
percent hits = 100.0
mean entropy = 1.4383
mean spherical score = 0.6369
point-biserial correlation of deltas = 0.8397
*
```

==== Posthoc ranking :

rank	weight	filename	pred:true	predval	meanval
1	42.7808	Muretus_Laud.txt	muretus + muretus	98.4375	55.6567
2	38.4468	Lauredan_FranVen.txt	sigonio ? neolats	91.4634	53.0166
3	26.4620	Lauredan_Mant.txt	sigonio ? neolats	59.7561	33.2941
4	23.2926	Sigonio_Dialogo.txt	sigonio + sigonio	59.7561	36.4635
5	19.6792	Muretus_Ingress.txt	muretus + muretus	35.9375	16.2583
6	16.8767	Cicero_Philippics7.txt	cics + cics	27.9255	11.0489
7	14.4874	ConsolA.txt	sigonio ? cons	37.8049	23.3175
8	14.2258	Cicero_Tusculan4.txt	cics + cics	57.1809	42.9551
9	12.3322	Seneca_Brevit.txt	cics ? claslats	26.3298	13.9976
10	11.3283	ConsolB.txt	sigonio ? cons	47.5610	36.2326
11	7.0514	EpistulaOct.txt	cics ? fake	11.9681	4.9167
12	5.8211	Abelard_HistCalamitatu	sigonio ? neolats	10.9756	5.1545
13	5.8211	Seneca_Otio.txt	sigonio ? claslats	10.9756	5.1545
14	4.1950	Heloise_Epistola_latli	sigonio ? neolats	8.5366	4.3415
15	2.8290	Seneca_Cons.txt	sigonio ? claslats	10.9756	8.1466
16	2.5690	Seneca_Prov.txt	sigonio ? claslats	6.0976	3.5285
17	1.8295	RhetHerr.txt	sigonio ? fake	3.6585	1.8290
18	0.5465	Seneca_Iral.txt	muretus ? claslats	1.5625	1.0160

+??++?+???????????

Inconsistent decisions :

Rankdif%	filename	predcats:truecat
0.00	Seneca_Cons.txt	muretus,sigonio:claslats
16.67	ConsolA.txt	muretus,sigonio:cons
22.22	Heloise_Epistola_latlib.tx	muretus,sigonio:neolats
22.22	RhetHerr.txt	cics,sigonio:fake
27.78	Seneca_Prov.txt	muretus,sigonio:claslats
33.33	ConsolB.txt	muretus,sigonio:cons
50.00	Seneca_Iral.txt	cics,muretus:claslats

The first few lines of this output simply echo some of the more important parameter settings from the input parameter file (cicsvocs.txt on the distribution). The rest of the output can be divided into three sections, delimited by the lines

==== Subsampling trial :

==== Holdout trial :

==== Posthoc ranking :

which mark results from the three phases of the program.

The first block (after a few header lines for identification purposes) displays the results of the subsampling trial. This takes the training corpus identified by trainmet and repeatedly splits it into 2 portions of size N and M. M is the rounded square root of the total number of texts in the training corpus and N is that total minus M, e.g. with 42 training files N will equal 36 and M will be 6. In each cycle, M texts will be picked at random and a "model" formed on the remaining N cases. Then that model will be used to predict the categories of each of the M texts absent from the model-building

procedure and the results recorded. This subsampling process continues until the total number of predictions made is at least 255. In the example above, that resulted in a total of 258 decisions. Only the first 42 of these are listed in detail, since there are only 42 individual files, but the confusion matrix and summary evaluation data is based on all 258 decisions.

Note that these 42 cases have been sorted in descending order of the column labelled "weight". This value is computed by simply taking the maximum model-match score and subtracting from it the arithmetic mean of all the matching scores. The higher this value the more clearly the predicted category's matching score exceeds that of the other categories. Thus items near the top of this list should indicate more confident decisions than those near the bottom, and we would expect more correct answers (marked with '+') near the top and more incorrect decisions (marked with '-') near the bottom.

The line

+++++-----++

that ends this list is just a string of these markers concatenated in order left to right from higher to lower. As expected, plus signs are more frequent towards the left side.

If there is a testmeta file, as there is in the above example, the next 2 blocks apply the models created from the training data to the holdout sample in 2 subtly different ways -- first as individual cases, i.e. just as in the subsampling phase, next with reference to the subsampling results as a whole, i.e. by trying to assess the extremity of each score in comparison with the scores obtained in phase 1. See next section....

Phase 5 : Interpreting the Output

In step (a) the program makes 258 decisions. It computes a matching score between each text and the category models (ensuring by subsampling that each case's data is excluded from its own category model) and, since the true category is known, considers the decision a success if the highest matching score is that of the true category.

At the foot of the subsampling block are evaluative statistics, not just raw success percentage, but a summary of the categorical decisions including a complete confusion matrix, which allows computation of recall and precision in each category. The measure to which I personally attach most importance is "point-biserial correlation of deltas". This is the point-biserial correlation of the differences (deltas) between each matching score and the average matching score for that text on the one hand and a binary (0/1) indicator which is 1 if the score concerned was for the true category and zero otherwise on the other hand. I regard this as a more sensitive measure of prediction quality than any measure based on the number of correct or incorrect decisions. This value has a maximum of 1, which would only come from perfect prediction.

The next block, beginning "====Holdout trial :", does essentially the same with the holdout sample, if one has been given. The confusion matrix may contain columns for categories not present in the training data, as in this case, where we have several "distractors". The program cannot determine whether it made a right or wrong decision in such cases, so they are marked with a question mark ('?'). Thus the line at the foot of this list of results

+?+?+?++???

indicates that the program could only make five definite decisions -- all correct as it happens and all in the left-hand half. (Would you expect me to pick a poor example?)

The third block, beginning "===Posthoc ranking :", is in my view the most interesting, but needs to be treated with caution. To illustrate, consider the results in this holdout sample, reproduced below.

==== Posthoc ranking :

rank	weight	filename	pred:true	predval	meanval
1	42.7808	Muretus_Laud.txt	muretus + muretus	98.4375	55.6567
2	38.4468	Lauredan_FranVen.txt	sigonio ? neolats	91.4634	53.0166
3	26.4620	Lauredan_Mant.txt	sigonio ? neolats	59.7561	33.2941
4	23.2926	Sigonio_Dialogo.txt	sigonio + sigonio	59.7561	36.4635
5	19.6792	Muretus_Ingress.txt	muretus + muretus	35.9375	16.2583
6	16.8767	Cicero_Philippics7.txt	cics + cics	27.9255	11.0489
7	14.4874	ConsolA.txt	sigonio ? cons	37.8049	23.3175
8	14.2258	Cicero_Tusculan4.txt	cics + cics	57.1809	42.9551
9	12.3322	Seneca_Brevit.txt	cics ? claslats	26.3298	13.9976
10	11.3283	ConsolB.txt	sigonio ? cons	47.5610	36.2326
11	7.0514	EpistulaOct.txt	cics ? fake	11.9681	4.9167
12	5.8211	Abelard_HistCalamitatu	sigonio ? neolats	10.9756	5.1545
13	5.8211	Seneca_Otio.txt	sigonio ? claslats	10.9756	5.1545
14	4.1950	Heloise_Epistola_latli	sigonio ? neolats	8.5366	4.3415
15	2.8290	Seneca_Cons.txt	sigonio ? claslats	10.9756	8.1466
16	2.5690	Seneca_Prov.txt	sigonio ? claslats	6.0976	3.5285
17	1.8295	RhetHerr.txt	sigonio ? fake	3.6585	1.8290
18	0.5465	Seneca_Iral.txt	muretus ? claslats	1.5625	1.0160

+????+?+???????????

Inconsistent decisions :

Rankdif%	filename	predcats:truecat
0.00	Seneca_Cons.txt	muretus,sigonio:claslats
16.67	ConsolA.txt	muretus,sigonio:cons
22.22	Heloise_Epistola_latlib.tx	muretus,sigonio:neolats
22.22	RhetHerr.txt	cics,sigonio:fake
27.78	Seneca_Prov.txt	muretus,sigonio:claslats
33.33	ConsolB.txt	muretus,sigonio:cons
50.00	Seneca_Iral.txt	cics,muretus:claslats

Here we have results from 18 cases unseen in the training phase, of which 11 are distractors, five are of known authorship and 2 (ConsolA and ConsolB) are the first and second halves of the purported *Consolatio Ciceronis* -- the item whose authorship motivated the collection of all this data.

Here predval isn't the maximum raw score obtained from matching against the category models: it is the maximum of relative scores, each relative score being the proportion of raw matching scores of the relevant category in the subsampling phase that this raw matching score exceeds, expressed as a percentage. Thus it can be regarded as a percentile. The "weight" is again its excess above the mean of all these percentiles.

Again the list is shown in descending order of weight, so the higher items should be more reliable than the lower ones. Satisfyingly, all the correct answers come in the upper half. The bottom half contains 8 of the eleven distractors along with ConsolB, the second part of the *Consolatio*, classed as by Sigonio. To take this line as an example (at rank 10) it shows that the matching score to Sigonio's model exceeded 47.5610% of the matching scores of Sigonio's known cases in the subsampling phase. This gets a weight of 11.3283 because it is that much larger than the mean of all 3 relative matching scores, which equals 36.2326.

Incidentally, two of the queried decisions in the top half of this list, at ranks 2 and 3, are cases in which the program categorized texts by Lauredanus as being by Sigonio. Lauredanus, pen name of Bernardino de Loredan, was Carlo Sigonio's student. In other words the system confused the pupil with his teacher. Given that it had no training examples of Lauredanus, this would seem a near-miss rather than an outright mistake.

The last portion of this output, labelled "Inconsistent decisions :", is intended to help deal with the

problem of distractors, i.e. documents from outside the known training categories. It does so simply by listing those cases where using the raw matching scores gives a different decision from using the relative matching scores. The listing is ordered by rankdiff% -- difference in ranks between the holdout and posthoc lists, expressed as percentage of the maximum possible rank difference. In this particular example it does relatively well. None of the seven cases listed is a correct answer: five are distractors and the other 2 are the halves of the disputed text, the Consolatio, which are split between Muretus and Sigonio. (In neither mode is either half assigned to Cicero, which tends to support suspicions of fakery.)

I believe this is an important aspect of the system. In text classification, as in other kinds of classification, the problem of never-encountered categories can loom large. (See, for instance, Eder, 2013.) Toccata, like most trainable classifiers, always picks the most likely category from those it has encountered in training, but the most likely may not be very likely; and accurately estimating just how likely, in a completely open set, is actually impossible. The percentiles give some useful information in this regard, and the list of inconsistent decisions backs that up. My personal rule of thumb is to take the posthoc listing, cross out anything with a weight of less than 10% as well as any items in the list of inconsistent decisions, and then take the remaining decisions as at least worthy of serious consideration.

There is no absolute answer, but these indications should be helpful to the human user, who will normally be using this sort of program in an exploratory context. Ultimately it will always be a matter of human judgement. My hope is that toccata can assist such judgements.

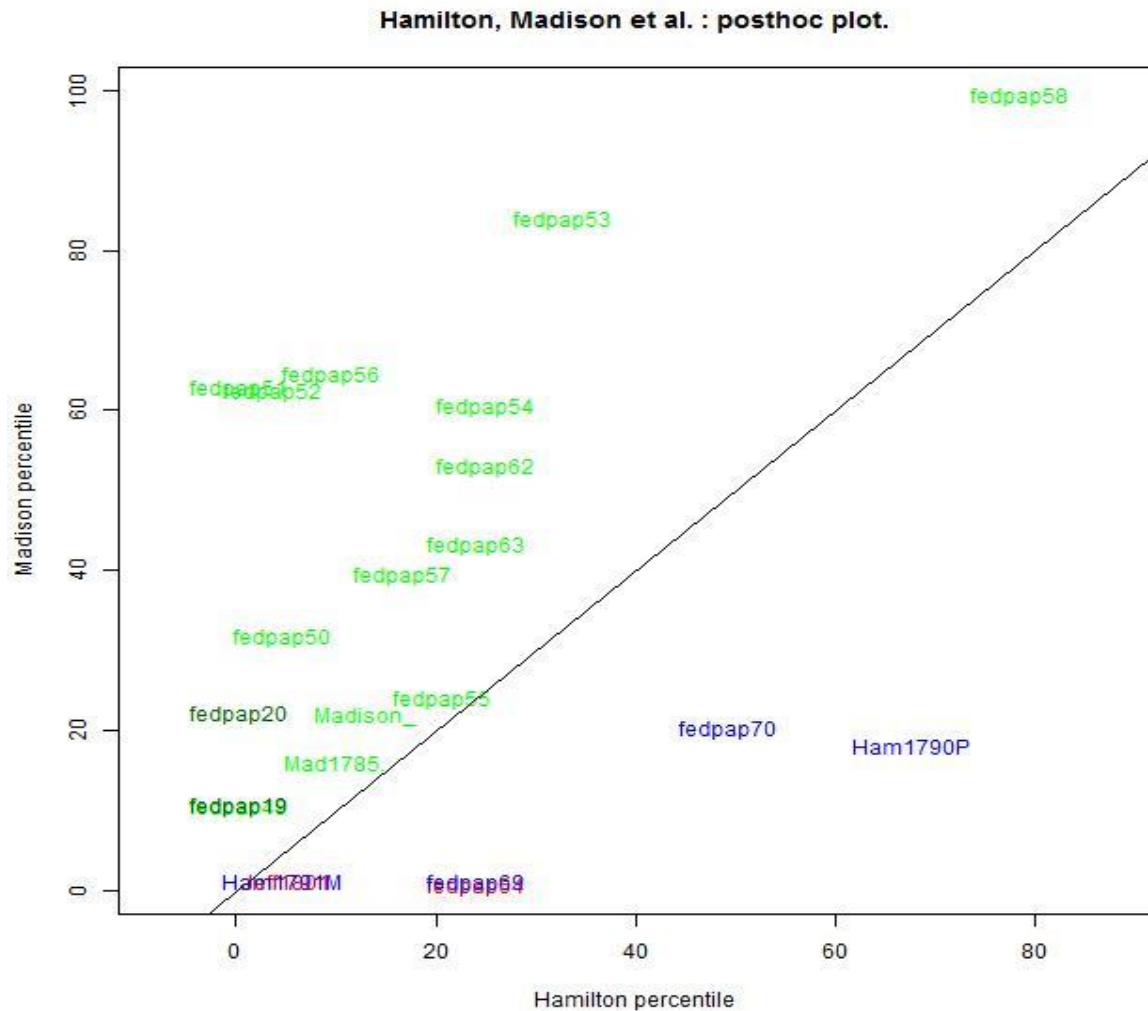
There's more

Running toccata7.py will produce four output files. The main listing (normally with base name ending "_list") is what has just been discussed. Two others will by default have base names ending with "_dump", "_mods". There will also be a file, simply called toccata.txt by default, with information of the system's parameter settings.

The _dump file is a tab-delimited .dat file intended to be imported into R for various statistical analyses. (It could also be imported into Excel, Minitab, SPSS et cetera.) The first five lines of the _dump file fedsvocs_dump.dat, which was produced by running toccata on the Federalist data, are listed below to illustrate the format.

mode	ok	textnum	filename	predcat	truecat	Hamilton	
						Madison	
testmode	+	3	fedpap08.txt	Hamilton	Hamilton	0.8535487	0.8365055
testmode	+	4	fedpap09.txt	Hamilton	Hamilton	0.8062115	0.7492547
testmode	+	28	fedpap36.txt	Hamilton	Hamilton	0.8466486	0.7153315
testmode	+	31	fedpap39.txt	Madison	Madison	0.6426143	0.7654597
....							

Essentially this file contains the results from all phases of the program (subsampling always, as well as holdout and posthoc, if a testmeta file is given) in a rectangular format that is acceptable to many statistical packages. The idea is that it allows further analyses, &/or graphical displays. For example, the plot below was generated in R from the posthoc decisions of a toccata run using the docalib_topvocs.py library. The jpeg file fedsvocs_postplot.jpg can be found on the op subfolder.



This plots the top 22 classifications out of 29 (thus ignoring the least-confident quarter according to posthoc ranking). The first eight characters of each file name are plotted, placed at the point defined by the Hamilton and Madison percentile scores. Hamilton texts are coloured blue; Madison texts, and the disputed papers (which since 1964 most people have believed to be authored by Madison) are in green; jointly authored Federalist papers are in dark green; and texts by other authors are in red.

Ideally a text by Madison should find itself towards the upper left corner of this graph (high scoring relative to known papers by Madison and low scoring relative to known papers by Hamilton). Conversely, works by Hamilton should appear towards the lower right side of this diagram. The black diagonal line indicates equal scores on both dimensions. All the green names are above this line and all the blue ones below it -- along with a couple of distractors (by Jay and Jefferson). This is consistent with the current opinion that Madison wrote all 12 disputed papers, and most of the joint papers. As it happens the disputed Federalist papers appear more Madisonian on this basis than the genuine works by Madison written before and after the Federalist.

Admittedly, including the 7 lowest-ranked texts would muddy this picture somewhat, mainly by causing an illegible splodge of lettering near zero on both axes. This raises a more general point. In some situations, a decision-maker is free to give "none of the above" as a response, in which case the posthoc ranking is genuinely valuable, since it allows dubious decisions to be avoided. However if a firm decision must be made in every case, then this doesn't help. (For fuller discussion of this issue, see Eder (2013).)

Mosteller and Wallace faced a situation in which the true author had to be one of Hamilton or Madison, but this kind of problem, with a small finite set of known candidate authors, is quite a rare luxury. More realistically, there is always some degree of uncertainty about whether the putative list of candidates does indeed include the true author. The possibility of joint authorship raises essentially the same issue. For instance, it is conceivable that Lauredanus assisted Sigonio in composing the 1583 Consolatio, in which case we wouldn't expect it to be very similar to works written by Sigonio alone. In this connection it is interesting that the 3 jointly written Federalist papers score low on both Hamilton and Madison dimensions: they aren't typical of either author.

And still more

As well as the _dump file, toccata7.py will produce a _mods output file. This contains the models generated from the whole training corpus, i.e. the models used to classify texts in the holdout and posthoc phases. Different methods will have models with different structures, so models from the four supplied libraries don't look the same. Yours, if you write a library module, will doubtless be different again. So there can't be a general guide to interpreting such models. Nevertheless, they usually will contain useful information. For instance, the model produced by running the docalib_maws.py library on the federalist corpus is, in effect, a keyword listing. Its first 7 entries are listed below.

```
classes = 2
docs = 64 64
vocsize = 96
48

id          toks2get=48 multivox=2
multivox    2
slug        2.0
toks2get    48
toksort     1

docfreq     63
hibayes     [0.37037037037037035, 0.8888888888888888]
highvals    [18, 14]
id          42
item        on
lobayes     [0.6296296296296297, 0.1111111111111111]
lowvals     [32, 0]
midrate     0.0038
newquay     0.2593

docfreq     60
hibayes     [0.6296296296296297, 0.1111111111111111]
highvals    [32, 0]
id          60
item        there
lobayes     [0.37037037037037035, 0.8888888888888888]
lowvals     [18, 14]
midrate     0.0025
newquay     0.2593

docfreq     53
hibayes     [0.6296296296296297, 0.1111111111111111]
highvals    [32, 0]
id          93
item        upon
lobayes     [0.37037037037037035, 0.8888888888888888]
lowvals     [18, 14]
midrate     0.0025
newquay     0.2593
```

```

docfreq      64
hibayes      [0.6111111111111112, 0.16666666666666666]
highvals     [31, 1]
id           1
item         to
lobayes      [0.3888888888888889, 0.8333333333333334]
lowvals      [19, 13]
midrate      0.0386
newquay      0.2222

docfreq      64
hibayes      [0.6111111111111112, 0.16666666666666666]
highvals     [31, 1]
id           11
item         at
lobayes      [0.3888888888888889, 0.8333333333333334]
lowvals      [19, 13]
midrate      0.0028
newquay      0.2222

docfreq      64
hibayes      [0.3888888888888889, 0.8333333333333334]
highvals     [19, 13]
id           22
item         and
lobayes      [0.6111111111111112, 0.16666666666666666]
lowvals      [31, 1]
midrate      0.0245
newquay      0.2222

docfreq      64
hibayes      [0.4074074074074074, 0.7777777777777778]
highvals     [20, 12]
id           9
item         by
lobayes      [0.5925925925925926, 0.22222222222222222]
lowvals      [30, 2]
midrate      0.0081
newquay      0.1852

```

This shows that the highest-ranked, most discriminatory, words for these 2 authors are: *on, there, upon, to, at, and & by*. If we look at the information associated with "to", we find that 31 of Hamilton's papers used "to" at a higher rate than the median for all 64 papers (3.86%) while in only 1 of Madison's 14 undisputed papers was "to" used with more than this relative frequency. If somebody writes and asks me to explain this in more detail, I might possibly agree.

It remains to point out that the `toccata.txt` file contains a list of all the program parameters and their values. Normally there is no need to look at this, but if a trial gives strange results it is sometimes useful to have a record of program settings.

Finally, is perhaps worth noting that using the same metafile as both `trainmet` and `testmeta` can sometimes be useful. From a strict classification point of view this is a kind of cheating, but the resulting holdout and posthoc listings may be informative. In effect they rank the texts by typicality. Thus they can be used to identify texts that are typical of their class (ranked near the top of the list) and those that are anomalous (ranked near the bottom) -- at least within the universe of discourse defined by the corpus as a whole.

References

Burrows, J.F. (1992). Not unless you ask nicely: the interpretive nexus between analysis and information. *Literary & Linguistic Computing*, 7(2), 91-109.

Eder, M. (2013). Bootstrapping Delta: a safety net in open-set authorship attribution. [*Digital Humanities 2013: Conference Abstracts*](#). Lincoln: University of Nebraska-Lincoln, 169-72.

Forsyth, R.S. (1995). *Stylistic Structures: a Computational Approach to Text Classification*. Unpublished Doctoral Thesis, Faculty of Science, University of Nottingham.
<http://www.richardsandesforsyth.net/doctoral.html>

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), 1-26.

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

Mosteller, F. & Wallace, D.L. (1984). *Applied Bayesian and Classical Inference: the Case of the Federalist Papers*. New York: Springer-Verlag.
[First edition, 1964.]

Appendix 1 : Metafiles

A metafile is a kind of data dictionary. It specifies which text files to work on, and may link associated data with each file. The main point is that metafiles can be read into a spreadsheet program such as Excel, modified, then written back out again to guide further processing (without necessarily rearranging a large collection of documents on disc). Another point to note is that all the software described herein assumes that the first 2 columns of a metafile are called "prepath" and "filename" and contain the file path then the file name. Columns within a metafile are delimited by the horizontal tab character. The toccata7.py program also needs a third column, called "doctype" by default.

The first line of a metafile is treated as a header, giving column names.

As an example, the Federalist training metafile (c:\toccata\samples\fed\mets\fed1.txt) is listed below.

prepath	filename	producer
c:\toccata\samples\fed\FedPaps\	fedpap01.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap06.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap07.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap08.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap09.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap10.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap11.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap12.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap13.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap14.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap15.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap16.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap17.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap21.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap22.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap23.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap24.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap25.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap26.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap27.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap28.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap29.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap30.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap31.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap32.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap33.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap34.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap35.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap36.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap37.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap38.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap39.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap40.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap41.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap42.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap43.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap44.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap45.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap46.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap47.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap48.txt	Madison
c:\toccata\samples\fed\FedPaps\	fedpap59.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap60.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap61.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap65.txt	Hamilton
c:\toccata\samples\fed\FedPaps\	fedpap66.txt	Hamilton

```

c:\toccata\samples\fedpaps\FedPaps\ fedpap67.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap68.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap70.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap71.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap72.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap73.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap74.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap75.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap76.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap77.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap78.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap79.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap80.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap81.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap82.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap83.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap84.txt Hamilton
c:\toccata\samples\fedpaps\FedPaps\ fedpap85.txt Hamilton

```

Here the category-column is called "producer" rather than "doctype", which would entail putting a line

targvar producer

into any parameter file using this metafile. (See Appendix 2.)

For this dataset, the corresponding holdout metafile (toccata\samples\fedpaps\holdout1.txt) is shown below. This contains works by some contemporaries as well as Hamilton and Madison. It also includes the disputed essays, coded as "Mad?".

```

prepath      filename      producer
c:\toccata\samples\fedpaps\holdout\ Ham1787PlanGovt.txt Hamilton
c:\toccata\samples\fedpaps\holdout\ Ham1790PublicCredit.txt Hamilton
c:\toccata\samples\fedpaps\holdout\ Ham1791ManuRept.txt Hamilton
c:\toccata\samples\fedpaps\holdout\ Jeff1801.txt Jefferson
c:\toccata\samples\fedpaps\holdout\ Lincoln1863Gettysburg.txt Lincoln
c:\toccata\samples\fedpaps\holdout\ Mad1785.txt Madison
c:\toccata\samples\fedpaps\holdout\ Madison_BillofRights_1789.txt Madison
c:\toccata\samples\fedpaps\holdout\ Mad1809.txt Madison
c:\toccata\samples\fedpaps\holdout\ Mad18151205.txt Madison
c:\toccata\samples\fedpaps\holdout\ fedpap04.txt JJay
c:\toccata\samples\fedpaps\holdout\ fedpap18.txt both
c:\toccata\samples\fedpaps\holdout\ fedpap19.txt both
c:\toccata\samples\fedpaps\holdout\ fedpap20.txt both
c:\toccata\samples\fedpaps\holdout\ fedpap49.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap50.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap51.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap52.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap53.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap54.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap55.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap56.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap57.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap58.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap62.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap63.txt Mad?
c:\toccata\samples\fedpaps\holdout\ fedpap64.txt JJay
c:\toccata\samples\fedpaps\holdout\ fedpap69.txt Hamilton
c:\toccata\samples\fedpaps\holdout\ fedpap70b.txt Hamilton
c:\toccata\samples\fedpaps\holdout\ sou1811.txt Madison

```

Of course, the point of metafiles is that they can be edited, so there is no need to stick to this particular selection.

minimet4.py

The easiest way to create an initial metafile is using the metaget.py file, described above under the heading "Phase 1". However, this uses the Tkinter library which seems to be sensitive to the exact version of Python 3 in use; so in case that doesn't work properly on your computer, I have left the more basic program minimet4.py in the distribution.

For example, to create a metafile for all the Federalist papers, the following parameter file could be supplied to minimet4.py.

```
comment    initial Federalist metafile :
jobname    fed0
corpath    c:\toccata\samples\fed\FedPaps\
metazero   c:\toccata\samples\fed\mets\fedzero.txt
targname    producer
targval    Hamilton
```

Briefly, corpath tells the program where the text files are located; metazero specifies the metafile to be created and where to place it; and targval gives the value to be put in the targname column. (More on parameter files below, in Appendix 2.)

Running minimet4.py with this parameter file (fed0.txt) would give the following output on screen.

```
>>>
C:\toccata\p3\minimet4.py 4.2 Thu Nov 28 16:06:37 2013
command-line args. = 1
prepath : C:\toccata\p3
working folder: C:\toccata\p3
script usage: python C:\toccata\p3\minimet4.py <parafilename>
please give parameter file name : fed0
Paths to search for parameter file :
['C:\\toccata\\parapath', 'C:\\toccata\\p3', '..', '. ',
'C:\\Users\\Richard\\parapath', 'C:\\Users\\Richard']
  fed0
trying to open : C:\toccata\parapath\fed0.txt
C:\toccata\parapath\fed0.txt opened for reading.
c:\toccata\samples\fed\mets
85 files read.
85 items written.

Output listing on : ..\op\minimeta.txt
Results dumped onto: c:\toccata\samples\fed\mets\fedzero.txt

C:\toccata\p3\minimet4.py done on Thu Nov 28 16:06:39 2013
after 0.25 seconds.
>>>
```

This would cause a metafile (fedzero.txt) to be placed on the c:\toccata\samples\fed\mets\ folder. The first five lines of this file are listed below.

```
prepath      filename      producer
c:\toccata\samples\fed\FedPaps\ fedpap01.txt Hamilton
c:\toccata\samples\fed\FedPaps\ fedpap02.txt Hamilton
c:\toccata\samples\fed\FedPaps\ fedpap03.txt Hamilton
c:\toccata\samples\fed\FedPaps\ fedpap04.txt Hamilton
....
```

You would have to edit this particular file, since it assigns all 85 texts to Hamilton, the majority author. However, a corrected metafile exists already (fed1.txt) so that isn't necessary in practice. (If

you are interested in exploring the case of the Federalist papers, a spreadsheet is provided (c:\toccata\samples\fedcs\metadat\fedcats.xls) that gives the categories of each of the 85 papers.)

Appendix 2 : Parameter Files

Parameters used by **toccata7.py**.

Parameter	Default value	Function
comment	[None]	This (or in fact any unrecognized parameter name, e.g. "##") can be used to insert reminders about what the file is meant to do.
atomize	1	This can be zero or 1. If it is 1, the input texts are tokenized by the program's built-in tokenizer. Only set this to zero if your files have already been tokenized, in which case whitespace will be considered to delimit tokens.
jobname	toccata	This gives the job a name. Any text string can be the value. It isn't necessary but it is useful as the jobname will be used as a prefix to the program's output files, so it can be seen that they form a group.
trainmet	[None]	This should be the full path specification of a metafile that indicates the text files that belong to the training corpus.
testmeta	[None]	This should be the full path specification of a metafile that indicates the holdout sample. It is optional: if omitted, the program only does the leave-n-out testing step.
wordonly	0	This should be integer 0 or 1. If it is 1, the tokenizer will ignore input tokens unless they begin with an alphanumeric character. If it is zero, all tokens will be considered, even sequences of punctuation symbols and so on. Unless you're sure the punctuation is original, it is advisable to set this parameter to 1.
casefold	1	This can be 0 or 1. Zero means that upper and lower case is left as found on input; 1 means that input texts will have all letters forced into lower case. (No effect on character sets without upper/lower case distinction.)
libname	docalib_topvocs	This should be the name either of one of the supplied classifier libraries (docalib_deltoid, docalib_keytoks, docalib_maws.py or docalib_topvocs.py) or a user-written library. Don't include the .py suffix, as this is appended automatically by Python.
paraline	[None]	This is an indirect way of passing parameters to the library, without having to rewrite the main toccata program. The format is to have items separated by spaces and to use the equal-sign '=' to separate the parameter name (left) from the parameter value (right). An example is paraline toks2get=48 multivox=2 which would tell the docalib_maws.py module to use the most discriminatory 48 from the most frequent 96 tokens in the training corpus. With docalib_topvocs, the only active parameter is corrmode: corrmode=ra specifies Spearman's rank correlation; any value other than 'ra' specifies Pearson's r. (More details below this table.)
randseed	1789	To ensure repeatability, Python's random number generator is initialized with this integer value. You can give a different random seed if you wish.
targvar	doctype	This should be the name of the column in the metafile(s) containing the class labels.
dumpfile	jobname with "_dump.dat"	The program dumps a rectangular file of the classification results in a form that is easy to import into R with the read.delim() function

	appended	for further processing. You can send this to a specific named file if you don't want to use the default name.
listfile	jobname with "_list.txt" appended	You can give a specific filename for the main output listing if you don't want it to have the default name.
modsfile	jobname with "_mods.txt" appended	This refers to a file where the classifier's decision models will be written.
outpath	subfolder "op" of current directory	You can send the output to a specified directory if you like.
outfile	toccata.txt (on outpath)	File where information on parameter settings will be written. (Really only needed for debugging.)

Library parameters given using paraline

Each prewritten library has a small number of internal parameters that can be set to non-standard values using toccata7's paraline parameter. It is important to note that resetting these values is optional. I have experimented to find sensible defaults, so the programs should work well without using paraline to alter the default settings. However, I know that users like to experiment, so brief descriptions are given below of how to change these values.

_deltoid

Here the only paraline parameter is topterms, which gives the number of (word-) tokens from the top of the ranked frequency list to employ as marker variables. For example,

```
paraline topterms=100
```

would cause the system to use the most frequent 100 words in the training corpus. If this parameter is absent, or outside the range 8 to 1024, the program will use the square root of the overall vocabulary size, which is usually a reasonable choice.

_keytoks

This has 2 adjustable parameters, snipsize and topkeys. For example,

```
paraline snipsize=256 topkeys=64
```

would tell the system to use snippets of size 256 tokens in its initial frequency/pervasiveness calculations, and keep the most 64 distinctive positive and negative keys (i.e. up to 128 tokens altogether) from each category as marker variables. Default snipsize is 115, the size of Shakespeare's 18th sonnet. If topkeys is not given or is outside the range 10 to 100, the square root of the overall vocabulary size is used.

_maws

This library has 2 adjustable parameters, toks2get and multivox. For example,

```
paraline toks2get=200 multivox=2
```

would instruct the system to pick the 400 (200 times 2) most frequent words (by document frequency) when building a model but retain only the 200 with the most apparent discriminatory effect, as measured by the variation in their above/below median usage rates across the text categories -- which can be regarded as a kind of keyness. Default values are toks2get=144 multivox=1.618034.

_topvocs

The only adjustable parameter for this library is corrmode, which specifies which type of correlation to use. For example,

```
paraline corrmode=pm
```

would cause the system to use Pearson's product-moment correlation in its similarity calculations. The default is equivalent to corrmode=ra, which causes the system to employ Spearman's rank correlation coefficient. In fact, any value other than 'ra' (or absence of this parameter) will cause the system to use Pearson's correlation. However, quite extensive testing suggests that rank correlation (the default) normally works better.

Parameters used by **minimet4.py**.

Parameter	Default value	Function
comment	[None]	This (or in fact any unrecognized parameter name, e.g. "##") can be used to insert reminders about what the file is meant to do.
corpath	[None]	Specification of directory where files to be included in metafile reside.
metazero	[None]	Full path/file specification of output metafile.
targval	00	Initial value to be given to the target column, normally a class label.
targname	doctype	Name to be given to the target column.
jobname	minimeta	This gives the job a name. Any text string can be used. It isn't necessary for this program.
outpath	[Subfolder "op" of current directory]	Directory where logging file will be written.
outfile	minimeta.txt	File where logging information will be written. (Really only needed for debugging.)

Normally only the first five in this table need to be specified by a user.

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

Note also that misspelt parameters are silently ignored!

Appendix 3 : Sample Screen Output

Below is roughly what you should expect to see on screen when running toccata7.py, in this case from the command prompt.

```
C:\2015>python c:\toccata\p3\toccata7.py
C:\toccata\p3\toccata7.py 7.0 Thu Jun 25 15:49:20 2015
command-line args. = 1
prepath : C:\toccata\p3
working folder: C:\2015
script usage: python C:\toccata\p3\toccata7.py <parafilename>
please give parameter file name : magskeys
Paths to search for parameter file :
['C:\\toccata\\parapath', 'C:\\\\toccata\\p3', 'C:\\2015', '..', '.',
'C:\\Users\\Richard.lounge-pc\\parapath', 'C:\\Users\\Richard.lounge-pc']
magskeys
trying to open : C:\toccata\parapath\magskeys.txt
C:\toccata\parapath\magskeys.txt opened for reading.
['prepath', 'filename', 'doctype']
144
target column name : doctype @ 2
Text types : {'maclearn', 'litling'}
Text-classifier s/w successfully loaded from library :
<module 'docalib_keytoks' from 'C:\\\\toccata\\p3\\docalib_keytoks.py'>
[Expected to contain definition of class Docadat]
Number of texts = 144
Number of tokens= 41501
Longest = 516 tokens.
Mean size = 288.2
Median size = 277.0
Smallest = 127
litling 75 21110
maclearn 69 20391
reference category : litling
snipsize = 115
Number of snippets = 343
total characters = 267478
255
477
1 12
477
2 24
485
3 36
483
4 48
477
5 60
475

[.... several similar lines omitted to save space ....]

17 204
477
18 216
482
19 228
479
20 240
478
21 252
473
22 264
unused test cases : []
264 trials.
```

Confusion matrix :

Truecat =	litling	maclearn
Predcat : litling	131	0
Predcat : maclearn	0	133

Kappa value = 1.0

Precision (%) by category :

litling 100.0

maclearn 100.0

Recall (%) by category :

litling 100.0

maclearn 100.0

cases = 264

cases with unseen category labels = 0

hits = 264

percent hits = 100.0

mean entropy = 0.7142

mean spherical score = 0.8386

point-biserial correlation of deltas = 0.905

490

Main output listed on : C:\toccata\op\mags_list.txt

Parameter settings on : C:\toccata\op\toccata.txt

C:\toccata\p3\toccata7.py done on Thu Jun 25 15:49:28 2015

after 4.96878 seconds.

It's nice to have an example of 100% correct classifications. Although these texts are short (median length 277 word tokens) this content-classification task is obviously easier than most realistic authorship problems.

Appendix 4 : Writing Your Own Classifier Library

To supply a bespoke classifier to `toccata7.py` you will have to provide a Python3 module with a class called `Docadat` that has at least the class methods listed below. The main program will supply your module with information through an object called `paradat`, which contains parameter values, and a list called `doclist`, which contains information derived from the texts. (More details below.)

```
def __init__(self,paradat,doclist):
```

This just creates an object holding the required data and methods. You are advised to copy the version in `docalib_maws` to begin with.

```
def loadpars (self,paradat,sep1=' ',sep2='='):
```

This interprets any parameters in `paradat.paraline` and stores them in `self.pars` (to avoid having the library alter values in `paradat`). Again, you might as well just copy this, and edit it to deal with any parameters that your system requires.

```
def makemods (self,paradat,doclist):
```

This should create the category models. (In fact it could be a single unified model, but the calling program still needs it to be called `makemods`.) In the supplied libraries, `makemods` always calls a method called `makemod` to create a model for each class one at a time. This seems tidy to me, but is not the only way.

```
def modprep (self,paradat):
```

This optional method will be called once, if present, before the subsampling process. The intention is to allow computations (e.g. on the whole training sample in `paradat.doclist`) which would be wasteful if repeated on every subsample in the subsampling trials. However, it is important not to 'cheat'; that is, information about the whole training set that should be invisible in the test subsets should not become available to the training subsets as a result of this method's operation.

```
def showmods (self,fo=sys.stdout):
```

This should be able to print a representation of the classification model/models.

```
def modsims (self,thisdoc,paradat):
```

This is the method that actually compares a document (`thisdoc`) with all category models and returns a matching score. Exactly how it achieves that will vary dependent on the technique implemented. In any case, it will have to return a list of numeric values, as many as there are categories in the training data (in the same order as in `paradat.catlist`). These are similarities, so the higher the value, the more closely that category matches the document. Estimated probabilities would do fine, though the scores don't have to be probabilities. Nor do they have to be positive. If your technique naturally produces distances, however, you will have to convert them to similarities somehow (e.g. as $-d$ or $1/(d+1)$).

You may well also have to write various internal service methods, depending on how your technique works, but the ones above are the necessary ones.

Yes, I admit it is a bit tricky, but the four libraries provided are liberally sprinkled with comments, so it should be possible for an experienced Pythoneer to compose a classifier that will be compatible with the `toccata` main program. The two major data structures that you need to know about, which `toccata7.py` supplies to the above `Docadat` methods are `doclist` and `paradat`.

doclist

This is a Python list, whose elements are Sack() objects. Sack is just a generic collection object. You can assume that each doclist element has the following attributes. (Your program can alter these, though that is most inadvisable!)

attribute	value
dnum	unique document id number, typically its position counting from zero
freqtab	a dictionary with word-tokens as keys and the frequencies of those word-tokens in the document as values (yes, that work is done for you!)
name	the name of the text file containing the document
outcome	the label of that document's category
size	the number of word-tokens in toklist
text	a space-delimited single string made by concatenating the items in toklist
toklist	a list of each (word-)token in the document, computed by my home-brew tokenizer if atomize=1 (the default) otherwise using white-space as a separator

paradat

This is a Sack() object which keeps together the main program's operational parameters. Again, it could be altered by the library methods, but in general that would be inadvisable. The attributes of paradat that you should be able to rely on are those described in the previous Appendix, as well as the following. If you run toccata7.py with one of the preexisting libraries and look at the toccata.txt output file you will see what other attributes might be of interest. Probably the only ones you'll need are those listed below.

attribute	value
catlist	a list of the category labels in the training corpus
cats	the number of different categories in the training corpus
docs	the number of documents in the training set
paraline	parameters specifically intended for the library, which can be unpacked by loadpars (into self.pars)

P.S.

I intend to add other library modules from time to time, and some day there may be a toccata8.py, which ideally will be an improvement on toccata7.py, by virtue of user feedback if there is any. I don't expect ever to reach version 10 though. :-)