TOCCATA : Text-Oriented Computational Classifier Applicable To Authorship

(User Notes by Richard Forsyth, April 2017)

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 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 Spitbol (an implementation of Snobol4). 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' with 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 6 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. Several 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, docalib_deltoid, docalib_keytoks, docalib_maws, docalib_tokspans, docalib_topvocs or docalib_vote 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 toccata9.py. (The digit is a version number, so may change as time goes by.) This performs three main functions, in sequence: (a) testmode: leave-n-out random resampling test of the classifier on the training corpus to provide statistics by which the classifier can be evaluated; (b) holdout: application of the classifier to an unseen holdout sample of texts, if a test metafile is given; (c) posthoc: re-application to the holdout sample of texts (if one is given) using the results from the testmode 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 "_dump" 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 (without markup, such as HTML tags). Each file is treated as an individual document, belonging to a particular category.

In the samples folder you will find 6 subfolders (ajps, bottlabs, cics, feds, mags and sonnets). 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, bottlabs, holds approximately 200 texts taken from the back labels of beer, wine and soft drinks bottles. It certainly isn't an authorship problem, but it is a useful test case since the texts themselves are relatively short. In addition to the three main categories -- beer, soft & wine -- there is a misc subfolder with a miscellany of other short text files. Some of these are cider back label texts, others from tea or coffee packaging, but some are not related to drinks at all and at least one isn't in English. Thus this collection, when trained to distinguish three categories can be used to assess how well Toccata deals with "distractors", i.e. texts that do not belong in any of the training classes.

The cics subfolder 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.

Lastly, the sonnets corpus contains 196 short English poems -- 14 sonnets by each of 14 different authors. This is a challenging problem firstly because the median length of each text in the training corpus is 116 words, secondly because 14 is a relatively large number of candidates. Hence the probability of successful classification by chance is just over 7 percent. There is also a holdout sample of 24 texts, absent from the training set. Half of these 24 items are 'distractors', i.e. texts by authors not present in the training set; 21 of these holdout texts are sonnets, but 3 are not: *Winter My Secret*, a poem of 239 words by Christina Rossetti; the short poem, *They Flee from Me*, by Thomas Wyatt, and Lincoln's 1863 Gettysburg address, which is the only example not in verse.

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
previous
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. Subfolder previous contains earlier versions of the software.

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 click on 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 click on 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. (I still write

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

There is also a program, randmets.py, which will take an input metafile and distribute entries at random to produce two output metafiles each of which contains a disjoint subset of the input metafile entries, randomly chosen. This is useful for producing training and test samples.

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 toccata9.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 toccata9.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 6 library modules exist already "off the shelf", to get you started:
docalib_deltoid, docalib_keytoks, docalib_maws.py, docalib_tokspans.py, docalib_topvocs.py and docalib_vote.

docalib_deltoid.py

This module 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 N 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 toccata9.py works with similarities, these mean dissimilarities (d_i) are converted to similarities as reciprocals, i.e. $1.0/d_i$. 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), rounded to an integer, which is usually a reasonable choice.

docalib_keytoks.py

This 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*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, capped at 256. 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.

docalib_maws.py
This 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.

docalib_tokspans.py
This method attempts to capture some of the information inherent in *syllexis*, the co-occurrence of words in close proximity, and especially sequential co-occurrence. Being primarily aimed at authorship attribution it starts, conventionally enough, by finding the most common words in the corpus. The number kept can be set by the user but by default is the square root of the vocabulary size, rounded to an integer, with an enforced maximum of 1024. Frequency is defined not by gross count but by the number of snippets in which a word occurs. Default snippet size is 144 tokens. Thus dispersed words that occur throughout the corpus are favoured over "bursty" words that occur often but only in few segments. This preliminary word-selection can be considered step 0 of the overall procedure, after which follow a further four steps: (1) accumulate; (2) eliminate; (3) correlate; (4) discriminate. Steps 1 and 2 constitute the model-building phase; steps 3 and 4 form the model-using phase.

Step 1 goes through the texts examining each segment of S consecutive words, where S is a parameter called spansize, set by the user. With spansize=3, a reasonable choice, the triplet "by the user", for example, would be examined and -- presuming "by" and "the" were in the frequent word list but "user" was not -- the tuple ("by","the") would be retained with 1 added to its occurrence tally. This typically generates a very large number of token tuples, most of which are shorter than spansize in length. In fact, the zero-length tuple is normally one of the most frequent -- implying a segment of spansize tokens all absent from the high-frequency list. Most of these tuples are winnowed out in step 2. In this step any such tuples occurring in fewer than 2 texts are removed, and, more significantly, the tuples are rated by how strongly they are associated with each category, with only the most distinctive retained. The default mode of calculating distinctiveness is to compute (r1-r2)*sqrt(df), where r1 is the rate in the category under consideration, r2 is the rate in all other categories combined and df is the number of documents of the category under consideration in which the token occurs. The highest-scoring items in each category are kept. The number to keep for each category is the rounded square root of that category's vocabulary size. The lists for each category are merged and become a list of distinctive features. In phase 3 this distinctive-feature list is used to calculate a correlation coefficient (rank correlation by default) of each test text with each

category in the following manner. The frequencies of every distinctive feature in the text under consideration, including zero, forms one vector which is correlated with vectors of the total frequencies of those features in each category, giving C correlations, where C is the number of categories. In step 4, the highest of those correlations is chosen to assign a category to the text concerned.

docalib_topvocs.py
This 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.)

docalib_vote.py
Module docalib_vote.py is exceptional in that it actually uses every single word-type in the training corpus as a feature. The 'model' developed for classification consists simply of frequency-tables for every text category, containing the frequency of that word in that category. To classify a new text, the frequency of every word in that text is counted; then, for each category, a similarity score is computed by adding a 'vote' by each word in the text to a running total. The vote is either sqrt(f) or 0, where f is the frequency of that word in the text being classified , according to whether the word's relative frequency is more common in the category concerned than in the corpus as a whole, or not. The category yielding the highest total is chosen. This simple procedure was intended merely to establish a baseline, but various trials have shown that it works rather well, perhaps because information from the whole vocabulary structure is utilized.

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.

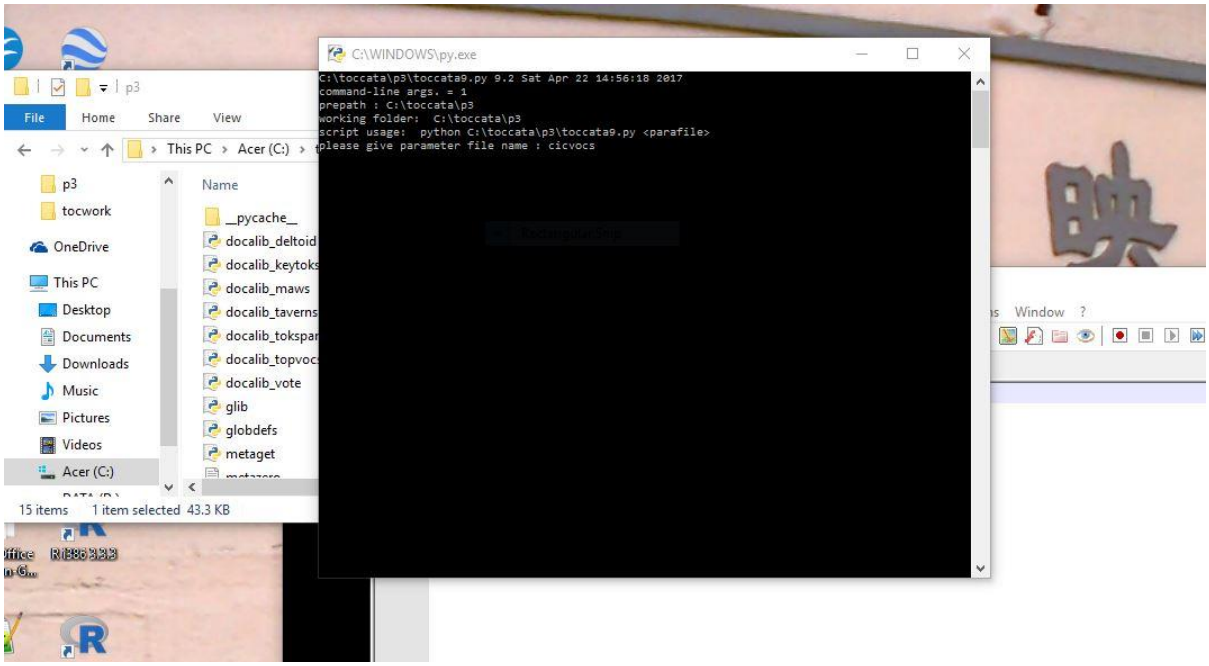
Parameter	Default value	Function
comment	[None]	This (or in fact any unrecognized parameter name, e.g. "##") can be used to insert reminders about what the file is meant to do.
jobname	toccata	This gives the job a name. Any text string can be the value. It isn't necessary but it is useful as the jobname 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, docalib_tokspans, docalib_topvocs.py or docalib_vote) 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 toccata9.py, for example with a command such as that below (shown in **bold**), you should see something like the following lines

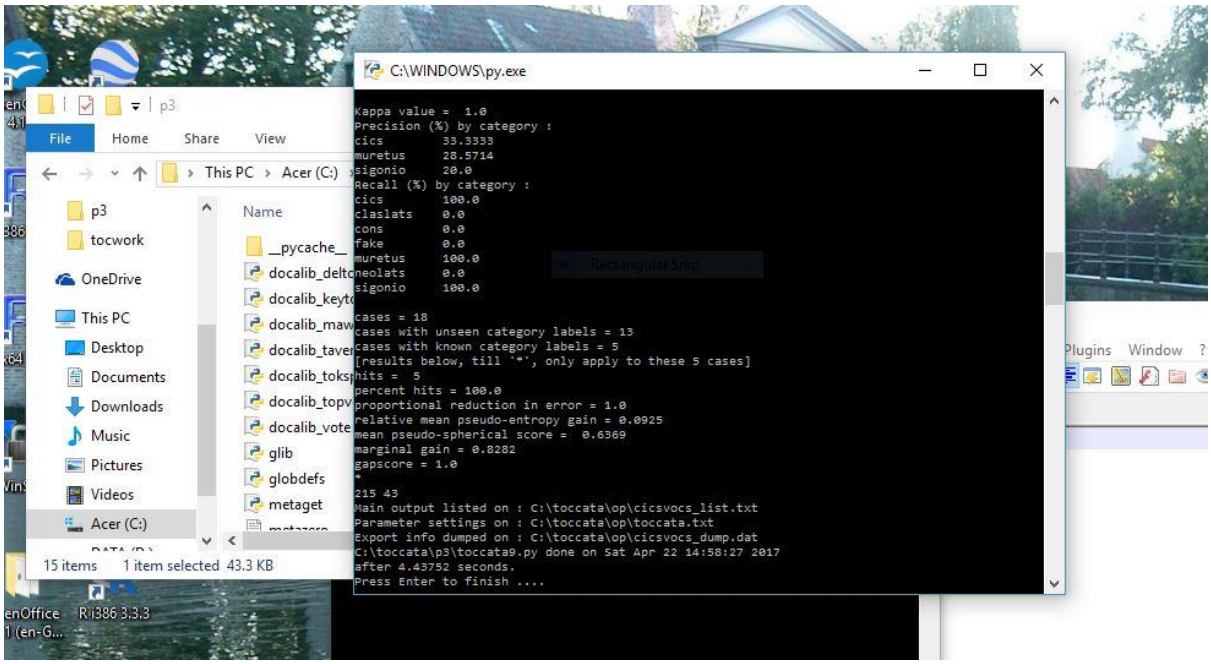
```
C:\2017> python c:\toccata\p3\toccata9.py
C:\toccata\p3\toccata9.py 9.2 Sat Apr 22 15:03:32 2017
command-line args. = 1
prepath : C:\toccata\p3
working folder:  C:\2017
script usage:  python C:\toccata\p3\toccata9.py <parafile>
please give parameter file name :
```

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).

More likely, you'll execute the program by double-clicking on toccata9.py in the toccata\p3 folder, which should produce a screen something like the example below.



Here the program has reached the point where the user is just about to press Return to select the cicsvocs parameter file. After pressing Return, various intermediate results will appear on screen, ending with something resembling the display shown below.



Using this parameter file, cicsvocs.txt, shown in the previous section on the cics dataset produces five 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 output files include a dump of data that could be exported into R or a comparable statistical package. More on them later.)

```
dateline  Sat Apr 22 14:58:18 2017
id        C:\toccata\parapath\cicsvocs.txt
libname   docalib_topvocs
progname  C:\toccata\p3\toccata9.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  meanrest
   1   0.3115  Cicero_Philippics2.txt          cics + cics         0.7923   0.4808
   2   0.3066  Cicero_Brutus1.txt              cics + cics         0.7230   0.4164
   3   0.2799  Cicero_Brutus2.txt              cics + cics         0.7650   0.4851
   4   0.2316  Cicero_Atticus1.txt             cics + cics         0.7841   0.5525
   5   0.2197  Cicero_ProQuinctio_latlib.      cics + cics         0.7062   0.4865
   6   0.1942  Cicero_InPisonem_latlib.tx      cics + cics         0.6734   0.4792
   7   0.1885  Cicero_ProFlacco_latlib.tx      cics + cics         0.7126   0.5241
   8   0.1881  Cicero_CatoSenectute.txt        cics + cics         0.7463   0.5581
   9   0.1727  Cicero_ProCaecina_latlib.t      cics + cics         0.7446   0.5719
  10   0.1720  Cicero_ProMarcello.txt          cics + cics         0.6793   0.5073
  11   0.1683  Cicero_ProSulla.txt             cics + cics         0.6660   0.4977
  12   0.1603  Sigonio_LaudHist.txt          sigonio + sigonio     0.7204   0.5602
  13   0.1562  Sigonio_LatLing.txt           sigonio + sigonio     0.7550   0.5988
  14   0.1541  Cicero_DeLegibus.txt            cics + cics         0.7782   0.6241
  15   0.1539  Cicero_ProMilone_latlib.tx      cics + cics         0.5992   0.4453
  16   0.1450  Cicero_Tusculan2.txt            cics + cics         0.7411   0.5961
  17   0.1407  Cicero_Amicitia.txt             cics + cics         0.7643   0.6236
  18   0.1316  Cicero_DeInventione2_latli    sigonio - cics        0.5994   0.4679
  19   0.1311  Sigonio_Elo1.txt              sigonio + sigonio     0.7060   0.5748
  20   0.1155  Cicero_InVerremII2_latlib.      cics + cics         0.7444   0.6289
  21   0.0957  Muretus_Pius.txt              muretus + muretus     0.6925   0.5967
  22   0.0912  Muretus_PaulFox.txt           muretus + muretus     0.6949   0.6037
  23   0.0899  Sigonio_Elo2.txt              sigonio + sigonio     0.6561   0.5662
  24   0.0895  Muretus_Phil.txt              sigonio - muretus     0.6788   0.5893
  25   0.0880  Cicero_Tusculan1.txt            cics + cics         0.7099   0.6218
  26   0.0860  Cicero_ProSexto.txt             cics + cics         0.6524   0.5663
  27   0.0852  Muretus_Rege.txt              muretus + muretus     0.6265   0.5413
  28   0.0845  Cicero_DePartitione_latlib      cics + cics         0.5574   0.4728
  29   0.0777  Cicero_DeFinibus.txt            cics + cics         0.6915   0.6138
  30   0.0754  Cicero_Orator.txt               cics + cics         0.6474   0.5719
  31   0.0584  Cicero_Officiis1.txt          sigonio - cics        0.6745   0.6161
  32   0.0581  Cicero_ProSestio_latlib.tx      cics + cics         0.6139   0.5558
  33   0.0581  Cicero_Somnium.txt              cics + cics         0.5967   0.5386
  34   0.0578  Muretus_Util.txt              muretus + muretus     0.7062   0.6484
  35   0.0554  Sigonio_HistIt4b.txt          sigonio + sigonio     0.6167   0.5613
  36   0.0514  Cicero_ProCluentio.txt          cics + cics         0.6068   0.5554
  37   0.0494  Sigonio_HistIt4a.txt          sigonio + sigonio     0.6154   0.5660
  38   0.0283  Cicero_ArchiaPoeta.txt        sigonio - cics        0.6859   0.6576
  39   0.0239  Cicero_Cat2.txt               sigonio - cics        0.5642   0.5404
  40   0.0231  Cicero_Rep2.txt               sigonio - cics        0.5948   0.5718
  41   0.0175  Cicero_DeImperio.txt            cics + cics         0.6430   0.6255
  42   0.0167  Cicero_NaturaDeorum2.txt      sigonio - cics        0.6543   0.6376
++++++++++++++++++-+++++-++++++-++++++---+-

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
proportional reduction in error = 0.5314
relative mean pseudo-entropy gain = 0.1034
mean pseudo-spherical score =  0.6441
marginal gain = 0.9174
gapscore = 0.9092

==== Holdout trial :

rank   weight  filename                          pred:true        predval  meanrest
   1   0.1472  Cicero_Philippics7.txt          cics + cics        0.6350   0.4878
   2   0.1458  EpistulaOct.txt                 cics ? fake        0.5916   0.4459
   3   0.1297  Muretus_Ingress.txt           muretus + muretus    0.6642   0.5345
   4   0.1200  Lauredan_FranVen.txt          sigonio ? neolats    0.7536   0.6336
   5   0.0908  Muretus_Laud.txt              muretus + muretus    0.7261   0.6353
   6   0.0740  Lauredan_Mant.txt             sigonio ? neolats    0.6927   0.6187
   7   0.0586  Sigonio_Dialogo.txt           sigonio + sigonio    0.6864   0.6278
   8   0.0558  Cicero_Tusculan4.txt            cics + cics        0.6935   0.6377
   9   0.0502  Seneca_Ira1.txt                 cics ? claslats    0.4870   0.4368
  10   0.0457  ConsolA.txt                   muretus ? cons       0.6425   0.5968
  11   0.0421  Seneca_Prov.txt               muretus ? claslats   0.5401   0.4980
  12   0.0418  Seneca_Otio.txt               sigonio ? claslats   0.5546   0.5128
  13   0.0413  RhetHerr.txt                    cics ? fake        0.4671   0.4258
  14   0.0395  Seneca_Brevit.txt               cics ? claslats    0.6248   0.5853
  15   0.0363  Seneca_Cons.txt               muretus ? claslats   0.6000   0.5638
  16   0.0318  ConsolB.txt                   muretus ? cons       0.6682   0.6363
  17   0.0199  Abelard_HistCalamitatum_la    sigonio ? neolats    0.5482   0.5282
  18   0.0181  Heloise_Epistola_latlib.tx    muretus ? neolats    0.5242   0.5061
+?+?+?++??????????

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
proportional reduction in error = 1.0
relative mean pseudo-entropy gain = 0.0925
mean pseudo-spherical score =  0.6369
```

```
marginal gain = 0.8282
gapscore = 1.0
*

==== Posthoc ranking :

rank credence filename                          pred:true      confidence congruity
   1   0.6749  Lauredan_FranVen.txt          sigonio ? neolats      0.4981   0.9146
   2   0.5783  Muretus_Laud.txt              muretus + muretus      0.3398   0.9844
   3   0.4304  Muretus_Ingress.txt           muretus + muretus      0.5154   0.3594
   4   0.4062  Cicero_Philippics7.txt           cics + cics         0.5907   0.2793
   5   0.3843  Lauredan_Mant.txt             sigonio ? neolats      0.2471   0.5976
   6   0.3293  Sigonio_Dialogo.txt           sigonio + sigonio      0.1815   0.5976
   7   0.2972  Cicero_Tusculan4.txt             cics + cics         0.1544   0.5718
   8   0.2650  EpistulaOct.txt                  cics ? fake         0.5869   0.1197
   9   0.1479  ConsolB.txt                   muretus ? cons         0.0560   0.3906
  10   0.1468  ConsolA.txt                   muretus ? cons         0.0811   0.2656
  11   0.1371  Seneca_Brevit.txt                cics ? claslats     0.0714   0.2633
  12   0.0885  Seneca_Otio.txt               sigonio ? claslats     0.0714   0.1098
  13   0.0546  Seneca_Cons.txt               muretus ? claslats     0.0637   0.0469
  14   0.0385  Abelard_HistCalamitatum_la    sigonio ? neolats      0.0135   0.1098
  15   0.0334  Seneca_Prov.txt               muretus ? claslats     0.0714   0.0156
  16   0.0163  Seneca_Ira1.txt                  cics ? claslats     0.1004   0.0027
  17   0.0138  RhetHerr.txt                     cics ? fake         0.0714   0.0027
  18   0.0135  Heloise_Epistola_latlib.tx    muretus ? neolats      0.0116   0.0156
?+++?++???????????
```

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 other 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.

There are also several other measures designed to assess the quality of the classification process. The Kappa value is Cohen's *kappa*, a multi-class index of agreement, computed according to the formula given in Siegel & Castellan (1988). There is also a proportional reduction in error measure, indicating how much the error rate is less than guessing based on the frequencies in each category.

The above measures are based on discrete outcomes, i.e. the integer number of correct or incorrect decisions. Some continuous indices are also printed. Arguably, these are more sensitive than measures based on whole numbers, as they are influenced by how much the correct category is rated above or below the other categories. The relative reduction in entropy and "spherical score" strictly only apply when the matching scores given to each category are intended as probabilities, thus only with docalib_maws among the supplied modules. However, rather than suppress them, the program makes an attempt to convert similarities to probabilities and attaches the prefix "pseudo-" to the entropy gain and spherical score. The latter is computed as p_c / √∑(p_i^2) where p_i is the probability or pseudo-probability of each category and p_c is the probability or pseudo-probability of the correct category.

More robust measures are "marginal gain" and "gapscore". Marginal gain is computed as goodgaps / allgaps where goodgaps is the sum of the differences between the similarity score of the system's chosen category and the mean similarities of the non-chosen categories in those cases where it was correct, and allgaps is the sum of these differences for all cases. Gapscore is intended as a parametric analogue of the proportional reduction in error statistic. The system records three values for each decision, the maximum similarity score, the mean similarity score and the similarity score assigned to the correct category. These are summed over all decisions as maxsum, meansum and truesum. Gapscore is then computed as

gapscore = (truesum - meansum) / ((maxsum - meansum)+tiny),

where tiny is 10 to the power of -16, just to avoid division by zero in degenerate cases such as when all scores are equal. A score of 1.0 will be attained if maxsum equals truesum, which happens if the system always assigns highest similarity to the correct category.

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 credence filename                          pred:true      confidence congruity
   1   0.6749  Lauredan_FranVen.txt          sigonio ? neolats      0.4981   0.9146
   2   0.5783  Muretus_Laud.txt              muretus + muretus      0.3398   0.9844
   3   0.4304  Muretus_Ingress.txt           muretus + muretus      0.5154   0.3594
   4   0.4062  Cicero_Philippics7.txt           cics + cics         0.5907   0.2793
   5   0.3843  Lauredan_Mant.txt             sigonio ? neolats      0.2471   0.5976
   6   0.3293  Sigonio_Dialogo.txt           sigonio + sigonio      0.1815   0.5976
   7   0.2972  Cicero_Tusculan4.txt             cics + cics         0.1544   0.5718
   8   0.2650  EpistulaOct.txt                  cics ? fake         0.5869   0.1197
   9   0.1479  ConsolB.txt                   muretus ? cons         0.0560   0.3906
  10   0.1468  ConsolA.txt                   muretus ? cons         0.0811   0.2656
  11   0.1371  Seneca_Brevit.txt                cics ? claslats     0.0714   0.2633
  12   0.0885  Seneca_Otio.txt               sigonio ? claslats     0.0714   0.1098
  13   0.0546  Seneca_Cons.txt               muretus ? claslats     0.0637   0.0469
  14   0.0385  Abelard_HistCalamitatum_la    sigonio ? neolats      0.0135   0.1098
  15   0.0334  Seneca_Prov.txt               muretus ? claslats     0.0714   0.0156
  16   0.0163  Seneca_Ira1.txt                  cics ? claslats     0.1004   0.0027
  17   0.0138  RhetHerr.txt                     cics ? fake         0.0714   0.0027
  18   0.0135  Heloise_Epistola_latlib.tx    muretus ? neolats      0.0116   0.0156
?+++?++???????????
```

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 questioned authorship motivated the collection of all this data.

The listing ranks the program's holdout decisions from most to least credible. The upper half includes all five correct assignments and four distractors. The lower half contains no correct answers, just nine distractors.

This output addresses the very real problem of documents from outside the known training categories. The listing is ordered by a quantity labelled "credence". This is simply the geometric mean of the last two numbers in each line, labelled "confidence" and "congruity". Confidence is derived from the preceding subsampling phase. To be specific, if W is the number of correct decisions with lower difference scores (labelled "weight" in the output listing) during the subsampling phase and L is the number of incorrect decisions with lower difference scores during that phase, then "confidence" is (W+L/2+0.5) / (S+1), where S is the number of subsampling trials. Congruity is computed as (0.5 + B) / (S+1), where B is the number of cases during the S subsampling trials in which items of the class selected had a lower similarity score to their own class model than that of the present instance. Thus congruity uses the randomized trials to estimate the empirical strength of similarity of the present case to its assigned category, while confidence estimates how the gap between the chosen category and the rest compares with those encountered during those trials.

This is an important aspect of the software. In text-classification, as with all kinds of classification, the problem of never-before-seen categories can loom large. (See, for instance, Eder, 2013.) Like

most trainable classifiers, Toccata 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 confidence and congruity scores give useful information in this regard. For example, all the bottom half (9 decisions) have both confidence and congruity scores less than 0.5, and none is correct. (We know that Muretus didn't write the *Consolatio*.) The list is shown in descending order. Satisfyingly, all the correct answers come in the upper half.

Incidentally, two of the queried decisions in the top half of this list, at ranks 1 and 5, 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.

There is no absolute answer to the "none-of-the-above" problem, 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.

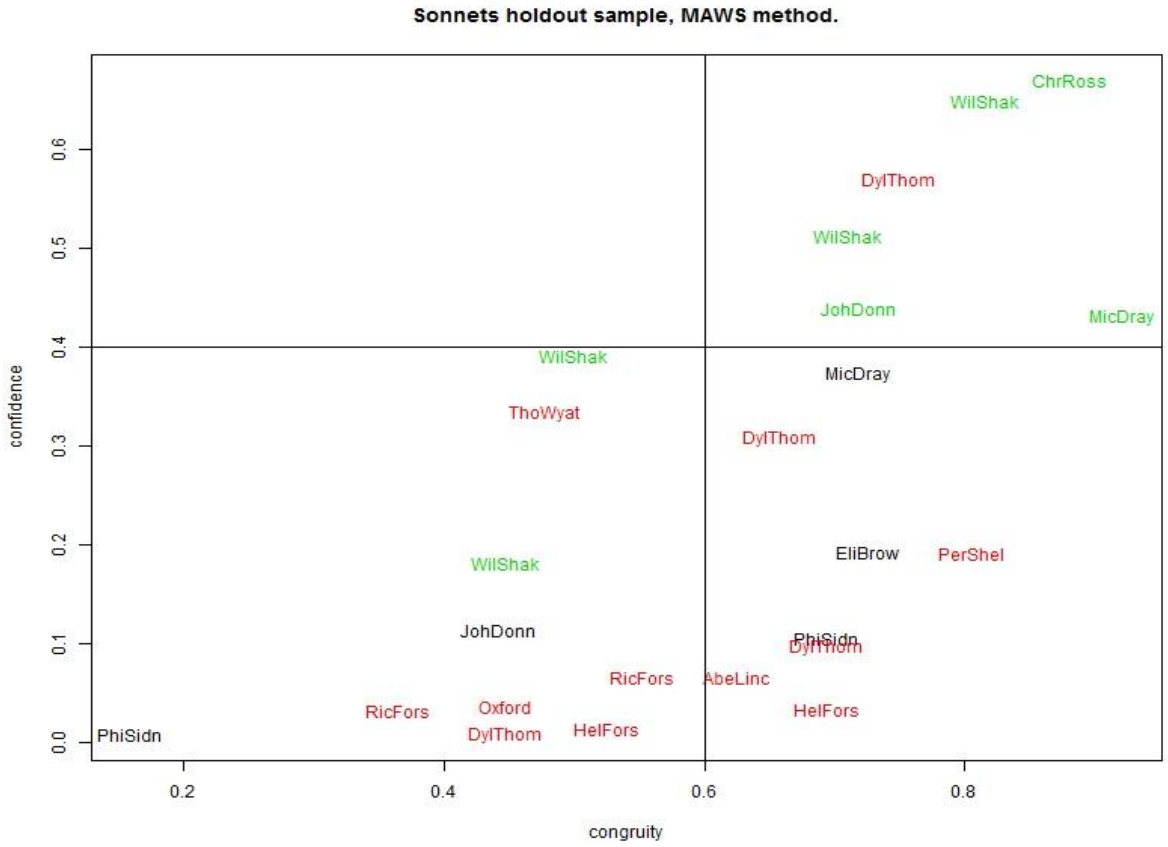
This is well illustrated by the following posthoc listing of the holdout data from the sonnets sample.

```
==== Posthoc ranking :

rank credence  filename                        pred:true     confidence congruity
   1   0.7685  ChrRoss_WinterSecret.txt      ChrRoss + ChrRoss      0.6704   0.8810
   2   0.7281  WilShak_6.txt                 WilShak + WilShak      0.6498   0.8158
   3   0.6534  DylThom_Altar09.txt           EdnMill ? DylThom      0.5693   0.7500
   4   0.6298  MicDray_Idea000.txt           MicDray + MicDray      0.4307   0.9211
   5   0.6038  WilShak_137.txt               WilShak + WilShak      0.5131   0.7105
   6   0.5624  JohDonn_Nativity.txt          JohDonn + JohDonn      0.4401   0.7188
   7   0.5175  MicDray_Idea048.txt           JohDonn - MicDray      0.3727   0.7188
   8   0.4509  DylThom_Altar05.txt           RupBroo ? DylThom      0.3090   0.6579
   9   0.4424  WilShak_109.txt               WilShak + WilShak      0.3914   0.5000
  10   0.3989  TomWyat_THEY_FLEE_FROM_ME.    EdmSpen ? ThoWyat      0.3333   0.4773
  11   0.3923  PerShel_Ozymandias.txt        EliBrow ? PerShel      0.1910   0.8056
  12   0.3740  EliBrow_SP23.txt              DanRoss - EliBrow      0.1929   0.7250
  13   0.2851  WilShak_RomeoJuliet.txt       WilShak + WilShak      0.1816   0.4474
  14   0.2699  PhiSidn_astel108.txt          EliBrow - PhiSidn      0.1049   0.6944
  15   0.2600  DylThom_Altar06.txt           EliBrow ? DylThom      0.0974   0.6944
  16   0.2248  JohDonn_Temple.txt            EdnMill - JohDonn      0.1142   0.4423
  17   0.2024  Lincoln1863Gettysburg.txt     SamDani ? AbeLinc      0.0655   0.6250
  18   0.1903  RicFors_LaBocca.txt           RupBroo ? RicFors      0.0655   0.5526
  19   0.1530  HelFors_1958.txt              EliBrow ? HelFors      0.0337   0.6944
  20   0.1262  oxford_13.txt                 WilWord ? Oxford       0.0356   0.4474
  21   0.1079  RicFors_Underworld.txt        EdnMill ? RicFors      0.0318   0.3654
  22   0.0830  HelFors_1982.txt              DanRoss ? HelFors      0.0131   0.5250
  23   0.0579  DylThom_Altar03.txt           RupBroo ? DylThom      0.0075   0.4474
  24   0.0345  PhiSidn_astel030.txt          EdmSpen - PhiSidn      0.0075   0.1591
++?+++-?+??-+-?-???????-
```

In this sample only half, 12 out of 24, of the texts come from the 14 categories in the training data. The other 12 are "distractors". Ranking by "credence" has done the job it is meant to do. In the upper half of the listing are 6 correct answers, 4 distractors and 2 mistakes; the lower half contains 1 correct answer, 8 distractors and 3 mistakes.

A graph, plotting these items in the dimensions of congruity and confidence, gives a visual impression of this result. In this plot correct decisions are coloured green, mistakes black and distractors red.



The upper right quadrant contains five correct decisions and one distractor. The lower left quadrant contains two correct decisions two mistakes and six distractors. In this example, confidence appears to be more informative than congruity, though this isn't always true.

Given that the median length of these texts is less than 120 words, that chance success would be 1 in 14, and that the training data for each known author consists of only about 1600 words, the system has done well to place 6 of the 7 correct answers in the upper half of its ranking and 8 of the 12 distractors in the lower half.

Mosteller and Wallace in 1964 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 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).)

There's more ....

Running toccata9.py will produce a number of 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.) To illustrate the format, the first five lines of the _dump file fedsvocs_dump.dat, which was produced by running toccata on the Federalist data, are listed below.

```
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.

Additionally, if a holdout sample is given, the program will produce a file of the same name as the _dump file, with _posthoc appended. This contains a tab-delimited version of the posthoc ranking, suitable for export to R and similar packages.

And still more ....

As well as the _dump file, toccata9.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 is no 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, 0.2222222222222222]
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.

Burrows, J.F. (2002). 'Delta': a measure of stylistic difference and a guide to likely authorship. *Literary & Linguistic Computing*, 17(3), 267-287.

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.]

Siegel, S. & Castellan, N.J. (1988). *Nonparametric Statistics for the Behavioural Sciences*. New York: McGraw-Hill.

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 toccata9.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\feds\mets\fed1.txt) is listed below.

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

```
c:\toccata\samples\feds\FedPaps\  fedpap67.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap68.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap70.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap71.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap72.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap73.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap74.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap75.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap76.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap77.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap78.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap79.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap80.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap81.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap82.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap83.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap84.txt  Hamilton
c:\toccata\samples\feds\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\feds\mets\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\feds\holdout\  Ham1787PlanGovt.txt Hamilton
c:\toccata\samples\feds\holdout\  Ham1790PublicCredit.txt    Hamilton
c:\toccata\samples\feds\holdout\  Ham1791ManuRept.txt Hamilton
c:\toccata\samples\feds\holdout\  Jeff1801.txt  Jefferson
c:\toccata\samples\feds\holdout\  Lincoln1863Gettysburg.txt  Lincoln
c:\toccata\samples\feds\holdout\  Mad1785.txt   Madison
c:\toccata\samples\feds\holdout\  Madison_BillofRights_1789.txt    Madison
c:\toccata\samples\feds\holdout\  Mad1809.txt   Madison
c:\toccata\samples\feds\holdout\  Mad18151205.txt     Madison
c:\toccata\samples\feds\holdout\  fedpap04.txt  JJay
c:\toccata\samples\feds\holdout\  fedpap18.txt  both
c:\toccata\samples\feds\holdout\  fedpap19.txt  both
c:\toccata\samples\feds\holdout\  fedpap20.txt  both
c:\toccata\samples\feds\holdout\  fedpap49.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap50.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap51.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap52.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap53.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap54.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap55.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap56.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap57.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap58.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap62.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap63.txt  Mad?
c:\toccata\samples\feds\holdout\  fedpap64.txt  JJay
c:\toccata\samples\feds\holdout\  fedpap69.txt  Hamilton
c:\toccata\samples\feds\holdout\  fedpap70b.txt Hamilton
c:\toccata\samples\feds\holdout\  sou1811.txt   Madison
c:\toccata\samples\feds\holdout\  PaineT_AgrarianJustice.txt TomPaine
```

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\feds\FedPaps\
metazero  c:\toccata\samples\feds\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 <parafile>
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\feds\mets
85 files read.
85 items written.

Output listing on : ..\op\minimeta.txt
Results dumped onto: c:\toccata\samples\feds\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\feds\mets\ folder. The first five lines of this file are listed below.

```
prepath      filename     producer
c:\toccata\samples\feds\FedPaps\  fedpap01.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap02.txt  Hamilton
c:\toccata\samples\feds\FedPaps\  fedpap03.txt  Hamilton
c:\toccata\samples\feds\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\feds\metadat\fedcats.xls) that gives the categories of each of the 85 papers.)

Appendix 2 : Parameter Files

Parameters used by **toccata9**.py. Note that misspelt parameters are silently ignored!

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, docalib_tokspans, docalib_topvocs or docalib_vote) 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.)
postcode	0	If zero, any _posthoc file will be written afresh, if postcode is 1 the posthoc results will be appended to the existing _posthoc file if it exists already.
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" appended	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 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 Toccata'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 8 to 256, 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.

_tokspans
This module uses parameters snipsize (default 144) as in docalib_keytoks, as well as spansize (default 3), spanmode (default 1) and topvocs (defaulting to the rounded square root of the vocabulary size if not given or outside the range 2 to 256). It also has a parameter spanmode (default 1). If spanmode is set to zero, the spans used will be treated as sets rather than tuples. This means that order is ignored, so that, for instance "of the" is not distinguished from "the of". When spanmode is zero, tokens within a span will be listed in alphabetic order on output, though this doesn't affect how they match.

_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.

_vote
This module does have a couple of parameters, flatfrex and rootfrex, but I believe they are best left as initialized by the software. They will probably disappear if there is an upgrade.

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.)

Appendix 3 : Sample Screen Output

Below is roughly what you should expect to see on screen when running toccata9.py, in this case from the command prompt.

```
C:\2017>python c:\toccata\p3\toccata9.py
C:\toccata\p3\toccata9.py 9.2 Mon Apr 24 12:42:22 2017
command-line args. = 1
prepath : c:\toccata\p3
working folder:  C:\2017
script usage:  python C:\toccata\p3\toccata9.py <parafile>
please give parameter file name : magskeys
Paths to search for parameter file :
['C:\\toccata\\parapath', 'C:\\toccata\\p3', '..', '.', 'C:\\Users\\Richard.lounge-
pc\\parapath', 'C:\\Users\\Richard.lounge-pc', 'C:\2017']
 magskeys
trying to open : C:\toccata\parapath\magskeys.txt
C:\toccata\parapath\magskeys.txt opened for reading.
?? Possible problem: no training file specified.
Using metafile c:\toccata\samples\mags\metadat\mag1.txt instead.
['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
total characters = 267478
snipsize = 115
Number of snippets = 343
255
1 12
2 24
3 36
4 48
.... [several similar lines omitted to save space] ....
18 216
19 228
20 240
21 252
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
proportional reduction in error = 1.0
relative mean pseudo-entropy gain = 0.2618
mean pseudo-spherical score =  0.8292
marginal gain = 1.0
gapscore = 1.0
Main output listed on : C:\toccata\op\mags_list.txt
Parameter settings on : C:\toccata\op\toccata.txt
Export info dumped on : C:\toccata\op\mags_dump.dat
C:\toccata\p3\toccata9.py done on Mon Apr 24 12:42:48 2017
after 4.32817 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 toccata 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 most of the supplied libraries, makemods 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 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 toccata9.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 metafile 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 toccata9.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
doclist	better not use this directly as it normally contains all training texts, not just the ones in the current main subsample!
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, but toccata9.py is probably the final numbered release. If I ever write a newer version it will just be toccata.py, and that really will be the last.