Methods Of Author Identification

Niamh McCombe
B.A. (Mod.) CSLL
Final Year Project, May 2002
Supervisor: Dr. Carl Vogel
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

__________________________
Niamh McCombe

June 16, 2002
Acknowledgements

Thanks to Dr. Carl Vogel, my project supervisor, for being so supportive and helpful,
to my friends for diverting me,
and to these women especially, for all their love and supportive, friendly vibes:
Barbara, Te, Lorna, Alice, Willow: thank you very much!
# Contents

1 **Introduction** 1

1.1 Aims ............................................. 2

1.2 Structure ........................................ 2

2 **Review of The Literature** 4

2.1 Introduction: Is there a Basis? ......................... 5

2.1.1 Dialect and Idiolect .......................... 5

2.1.2 Language as an automatic process .............. 5

2.1.3 Caveat: Metalinguistic awareness ................. 6

2.2 Examples of linguistic features to be used for author identification ............................................ 6

2.2.1 Variant structures of prepositional phrases .... 6

2.3 Tests of potential author-identifying features: the Chaski report 6

2.3.1 Type/Token Ratio ................................ 7

2.3.2 Hapax Legomena Ratio ............................. 7

2.3.3 Readability Measures ............................... 8

2.3.4 Content Analysis .................................. 8

2.3.5 Spelling Errors .................................... 8

2.3.6 Prescriptive Grammatical Errors .................. 9

2.3.7 Sentential complexity .............................. 9

2.3.8 Syntactically-classified punctuation ............... 10

2.3.9 Abstract syntactic structures ..................... 10

2.3.10 Chaski’s conclusions ............................ 10

2.4 A response to Chaski ................................. 10

2.4.1 History of Stylometry/Authorship Attribution .... 11

2.4.2 The Criticisms of Chaski .......................... 11

2.4.3 Grant and Baker’s Conclusions .................... 12
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>4.5.3 Discussion of Algorithm</td>
</tr>
<tr>
<td>35</td>
<td>4.6 The Symmetrical Algorithm for calculating Corpus Similarity Scores using Chi By Degrees of Freedom</td>
</tr>
<tr>
<td>36</td>
<td>4.7 The &quot;Ratio&quot; Method for determining which Program Specifications are most likely to lead to successful Author Identification</td>
</tr>
<tr>
<td>36</td>
<td>4.8 Introduction</td>
</tr>
<tr>
<td>37</td>
<td>4.9 Algorithm for the Ratio Method</td>
</tr>
<tr>
<td>38</td>
<td>5 The Concordancers</td>
</tr>
<tr>
<td>39</td>
<td>5.1 Introduction</td>
</tr>
<tr>
<td>39</td>
<td>5.2 concordance</td>
</tr>
<tr>
<td>39</td>
<td>5.2.1 Command Line Options in concordance</td>
</tr>
<tr>
<td>39</td>
<td>5.3 punconcord</td>
</tr>
<tr>
<td>39</td>
<td>5.3.1 Command Line Options in punconcord</td>
</tr>
<tr>
<td>40</td>
<td>5.4 letter</td>
</tr>
<tr>
<td>40</td>
<td>5.4.1 Command Line Options in letter</td>
</tr>
<tr>
<td>40</td>
<td>5.5 tagconc.pl</td>
</tr>
<tr>
<td>40</td>
<td>5.5.1 Command Line Options in tagconc.pl</td>
</tr>
<tr>
<td>40</td>
<td>5.6 keywords</td>
</tr>
<tr>
<td>41</td>
<td>5.6.1 Command Line Options in keywords</td>
</tr>
<tr>
<td>43</td>
<td>6 About the Tagger</td>
</tr>
<tr>
<td>43</td>
<td>6.1 The Tagger</td>
</tr>
<tr>
<td>43</td>
<td>6.2 postprocess.pl</td>
</tr>
<tr>
<td>43</td>
<td>6.3 Relevant filenames</td>
</tr>
<tr>
<td>44</td>
<td>7 How To Use the Programs</td>
</tr>
<tr>
<td>45</td>
<td>7.1 chisquare</td>
</tr>
<tr>
<td>46</td>
<td>7.2 multicdf</td>
</tr>
<tr>
<td>46</td>
<td>7.2.1 Command line options for multicdf</td>
</tr>
<tr>
<td>48</td>
<td>7.2.2 General assumptions made by multicdf</td>
</tr>
<tr>
<td>48</td>
<td>7.2.3 Sample Output And Input</td>
</tr>
<tr>
<td>49</td>
<td>7.2.4 Command Line Options for multicdf.new</td>
</tr>
<tr>
<td>49</td>
<td>7.2.5 Assumptions made by multicdf.new</td>
</tr>
<tr>
<td>49</td>
<td>7.3 interface</td>
</tr>
<tr>
<td>50</td>
<td>7.3.1 Command line options for interface</td>
</tr>
<tr>
<td>54</td>
<td>7.3.2 General assumptions made by interface</td>
</tr>
<tr>
<td>55</td>
<td>7.4 mean</td>
</tr>
</tbody>
</table>
CONTENTS

7.4.1 Command Line Switches in **mean** .......................... 55
7.4.2 Examples of Input and Output ................................. 57
7.5 An Example Program Run ........................................ 57
  7.5.1 Corpora ................................................. 57
  7.5.2 Corpus File ............................................ 57
  7.5.3 Running **interface** ..................................... 58
  7.5.4 Creating the list of **.rank** files ......................... 61
  7.5.5 Running **mean** ........................................ 61
7.6 Deciphering The Filenames ....................................... 62
  7.6.1 Concordanced Files ....................................... 62
  7.6.2 Lists of Concordanced Files ............................... 64
  7.6.3 Results Files ........................................... 65

8 Results and Discussion ........................................... 68
  8.1 Which Ngrams are likely to be significant? ..................... 69
    8.1.1 Corpora ............................................. 69
    8.1.2 Which single letters were the most significantly different? 69
    8.1.3 Which bigrams were the most significantly different? ... 70
    8.1.4 Which words were the most significantly different? ...... 70
    8.1.5 Which tag bigrams were the most significantly different? 71
    8.1.6 Is it possible to identify the author on the basis of this kind of evidence? .................................. 72
    8.1.7 What, if any is the relation between “obvious” and “useful” methods of author identification?” ................... 72
  8.2 Which of the tested methods are the most effective means of author identification? ........................................ 73
    8.2.1 The corpora ........................................... 73
    8.2.2 The Input ............................................. 73
    8.2.3 The Task .............................................. 73
    8.2.4 The Results ........................................... 74

9 Conclusion ..................................................... 80
  9.1 On the possibility of author identification ...................... 81
  9.2 The software .............................................. 81
  9.3 The most effective methods ................................... 81
  9.4 Implications ................................................ 81
  9.5 Other Effective Methods ..................................... 82
  9.6 Possibilities for further work ................................ 82
CONTENTS

10 Appendix 1: Code for Concordancers 85
11 Appendix 2: Code for Other Programs 93
12 Appendix 3: Full results of LJ experiment 103
List of Figures

2.1 Example Cusum graph ........................................ 15
2.2 Example Cusum graph ........................................ 16
2.3 Example Cusum graph ........................................ 17
2.4 Example Cusum graph ........................................ 18

4.1 Example of a table for calculating chi square values .... 28
4.2 Example of a table for calculating chi square values, with expected frequencies .......................... 30
4.3 Example of a table for calculating chi square values, with expected frequencies and chi square values ........ 30

7.1 flowchart for program interface .............................. 51
Abstract

A variety of methods of author identification are examined, in the literature and through a specially developed suite of programs. It is found that the programs developed are extremely effective at discriminating authors in the corpora that were examined, particularly when frequencies of unigrams of characters are used as the discriminant factors.
Chapter 1

Introduction
1.1 Aims

The aim of this project is to investigate methods of computer-based author identification, to determine which methods of author identification are most effective, and to create a computer program or set of computer programs that can identify the author of a given text.

1.2 Structure

This paper is divided into the following chapters:

Introduction

You’re reading it.

Review of the Literature

A close look at some of the literature that has been written about methods of author identification, and a description of the techniques involved.

Methods Chosen for Further Investigation

This section describes the methods of author identification that were chosen as the focus of this project.

Algorithms and Mathematical Methods

This section describes the algorithms and methods used in this project, including the chi square technique, and the algorithms for the suite of programs.

The Concordancers

A description of, and instructions for, the concordancing programs that were created for the purposes of this project.

The Tagger

Some information about the tagger that this project makes use of.
 CHAPTER 1.  INTRODUCTION  3

How to Use the Programs
Deatailed instructions on how to use the programs in this suite, and what all the command line options mean.

Results and Discussion
Discussion of which methods of author identification proved to be most successful, using the results from these programs

Conclusion
What was learned/achieved in the course of this project, and avenues for further investigation.
Chapter 2

Review of The Literature
2.1 Introduction: Is there a Basis?

Chaski, in [1], poses the fundamental question: is there a theoretical basis for language based author identification within the science of linguistics? Her answer is that there is, based on the concepts of dialect and idiolect, and automatic language processing.

2.1.1 Dialect and Idiolect

Dialect is a regional variation within language: idiolect is an individual’s variation on the dialect. She offers the example of Southern American English speakers who will use the formulation

\[ \text{John might should check his parking meter.} \]

While it is possible to identify a speaker as a Southern American by his use of this formulation (dialect), it is not possible to say that Southern American English speakers are homogenous in their use of it. One Southerner might only produce this formulation in statements, while another might also produce questions such as

\[ \text{Should you might check the meter?} \]

This kind of individual variation is idiolect.

2.1.2 Language as an automatic process

In normal language processing, the construction of syntax is unconscious. Most people remember the meaning(or semantics) of what someone has said long after they forget the syntax (or form). The syntax of language is a highly complex set of rules which most of us are completely unconsciously aware of. It is generally accepted among linguists that the syntax of language is generated automatically and unconsciously, so that we can focus on meaning as we communicate.

The fact of language being an automatic process gives us some scope for linguistic fingerprinting: the more automatic a process is, the less control we have over it and the greater its reliability as an indicator of individuality.
2.1.3 Caveat: Metalinguistic awareness

It is possible to distance ourselves from language and analyse it consciously (otherwise there would be no science of linguistics at all). There is no theoretical basis for assuming that it is impossible for someone to have such an extreme degree of metalinguistic awareness, as to be capable of completely altering their own idiolect, mimicking the idiolecting patterns of another speaker and suppressing their own. The theoretical position must be taken that linguistic disguise is possible, depending on the author’s level of metalinguistic awareness.

2.2 Examples of linguistic features to be used for author identification

The literature here suggests many, many possible alternatives. Chaski followed her discussion of the theoretical basis in [1] with a description of some features that exhibited an "identifying ability". Clauses per sentence and phrases per sentence were found to have identifying ability.

2.2.1 Variant structures of prepositional phrases

The most common form of prepositional phrase is

\[ PP \rightarrow P \ NP \]

The ratio between this form of prepositional phrase and variants on the form was found both to identify a particular subject’s idiolect, and to remain constant across her samples: hence, it both identifies and differentiates her language and is a valid marker. This marker was found to be valid even across small writing samples.

2.3 Tests of potential author-identifying features: the Chaski report

Chaski, in [2] starts by drawing a distinction between scientific, replicable methods of author identification and the kind of "literary criticism" methods used by Donald Foster (see [3]) The difference is that scientific methods are
based on empirical, testable hypotheses, and the use of these methods can be done by anyone: it is not dependent on a special talent or knack.

She gives us nine empirical hypotheses that have been used to identify authors in the past, and which she proposes to test:

- Vocabulary richness identifies authors
- Hapax Legomena identifies authors
- Readability measures identify authors
- Content Analysis identifies authors
- Spelling errors identify authors
- Grammatical errors identify authors
- Syntactically classified punctuation discriminates between authors
- Sentential complexity identifies authors
- Abstract syntactic structures differentiate and identify authors.

Her first step in testing these hypotheses was to assemble a writing sample database. She then extracted four selections of samples from test subjects, using collections of short samples from women of similar ages and levels of education.

2.3.1 Type/Token Ratio

This is the ratio of the number of distinct words (type) to the number of total words (token): it is considered to be a measure of vocabulary richness. It was found using a t-test that Type/Token Ratio clustered the texts into two groups and not four, and failed in this test to discriminate between authors.

2.3.2 Hapax Legomena Ratio

This is the ratio of the number of words occurring once (Hapax Legomena) to the total number of words. It was found using a t-test that this ratio clustered the authors into three groups rather than four, and also misclassified one of the samples, and failed in this test.
2.3.3 Readability Measures

Readability measures calculate the supposed complexity of a document, and are calculations based on sentence length and word length. See [4] for more information on the calculations used to obtain readability scores.

It was found that readability scores had high correlations across all writing samples written by the same author. Unfortunately these correlations did not decrease when the unknown writing sample was grouped with each of the different writers.

The t-test also failed to discriminate authors based on their readability scores. Overall, the readability scores were deemed useless as a method of author identification.

2.3.4 Content Analysis

The basis of this method was to classify each word in the document by semantic category, and statistically analyse the distance between documents. The specific semantic categories used are not fully explained in Chaski’s paper or anywhere in the bibliography; however since even the inventor of the semantic classification system involved, McTavish, does not believe this system is useful as a method of discriminating between authors, and it failed this test, there is little point in examining this technique further.

2.3.5 Spelling Errors

Simple and sophisticated techniques of analysing spelling errors were looked at. One approach was to subclassify the spelling errors into such categories as

- errors with doubled consonants
- errors with doubled consonants with suffix
- errors with vowels preceding nasal consonants
- errors with [I] (vowel) sound

None of the techniques or classifications tried resulted in discrimination between authors.
2.3.6 Prescriptive Grammatical Errors

The prescriptive grammatical errors tested were:

- sentence fragment
- run-on sentence
- subject-verb mismatch
- tense shift
- wrong verb form
- missing verb

Analysing errors by type led to six clusters of authors; however, while the number of clusters was wrong, the contents of each cluster were correct—ie no samples from different authors were mistakenly classified together. Also the unknown sample was correctly assigned.

Analysing errors by frequency led to no useful result.

Chaski points out that since this kind of prescriptively defined grammatical error is a function of many nonstandard dialects, so to use this as a discriminator confounds class with individuality; also that it is statistically hard to quantify this kind of error, and also that this kind of technique has no basis whatsoever in modern linguistic theory. So even though partial success was achieved the report does not endorse this technique.

2.3.7 Sentential complexity

Chaski says ”The underlying principle in sentential complexity analysis is the idea that some sentence structures are more complex than others and that people will differ in their abilities to produce different types of sentential complexity” The sentences produced by the writers were classified by complexity, and analysed using the chi-square test. This test only discriminated strongly between two authors, leaving the others as one large cluster. However Chaski points out that different ways of measuring sentential complexity might have led to different, better results.
2.3.8 Syntactically-classified punctuation

Each punctuation mark is listed and classified by syntactic function—e.g. End-of-Sentence period, comma separating main and dependent clauses, comma in list... Frequencies are assigned to each type of punctuation.

Chaski says “The underlying principle in punctuation analysis is the idea that punctuation reflects intonation, which is driven by syntactic structure”

The chi-square test is used to analyse the data. In this task, the frequency of syntactically classified punctuation patterns was found to discriminate the author successfully, and assign the unknown sample to the correct author. This method is endorsed by the report.

2.3.9 Abstract syntactic structures

Here Chaski refers again to the system she described in [1], which computationally analyses syntactic patterns. She uses verb phrase structure as a differentiating feature, and, once again she cites the ratio of prepositional phrase structures as both a differentiating and identifying feature. This ratio was tested using chi-square on its ability to both identify and distinguish authors, and once again was found to perform on both counts. This method is endorsed by the report.

2.3.10 Chaski’s conclusions

Chaski finds that methods which are based on ”common misconceptions about language” and a layperson’s understanding of language, are unreliable. The two methods which rely on linguistic science/generative grammar appear to accurately cluster and distinguish documents.

2.4 A response to Chaski

In [5], Grant and Baker provide a brief history of stylometry and author identification as well as a criticism of Chaski’s report on methodological grounds.


2.4.1 History of Stylometry/Authorship Attribution

Authorship attribution studies have historically been concerned with literary, historic and religious texts. Stylometry probably originates with the suggestion of Augustus de Morgan in 1851 that it might be possible to identify biblical authors because "one might deal in longer words" [6], quoted in [5].

The groundwork for the modern era in authorship attribution was laid by Mosteller and Wallace in their 1968 study of the *Federalist Papers* which is cited in a great deal of the literature. This is a set of 146 eighteenth century essays, written by three different authors. In the vast majority of cases the author is known: the authorship of twelve, however, is disputed. Mosteller and Wallace used a Bayesian analysis of ninety function words and sixty other words to give odds for assigning each text to a given author.

The next most common measures used are vocabulary richness and repetition—the Type-Token and Hapax Legomena methods described above. Word type frequencies and distributions, syntactic analysis, co-occurrence and collocations, word, sentence clause and paragraph lengths and distributions, and content analysis.

Multivariate approaches are frequently used, as there are so many possible discriminants.

Grant and Baker describe the approach known as Principal Component Analysis which identifies which marker or combination of markers discriminates in a particular case (my italics) as "particularly effective", and their report heavily endorses this approach.

2.4.2 The Criticisms of Chaski

Grant and Baker’s essential concern is that just because a particular marker has been shown to discriminate in a particular set of texts, there is no reason to assume that it will discriminate either between any texts written by authors A and B, or that this particular marker is in general a "good" marker of authorship.

They point out that the fact that Chaski uses a deliberately limited sample, in order to ensure that her subjects are close in age and socioeconomic background, has a negative effect on the extent to which her conclusions might be assumed to be generalisable to a wider population.

They make a case that no marker can be said to be valid, unless there is a solid theoretical construct behind it, and a hypothesis about why this
marker might be a useful discriminator between individuals.

They also claim that Chaski chooses to accept the null hypothesis of two results being from the same population, where the significance level is not above threshold, whereas a more accurate conclusion to draw is that the null hypothesis cannot be rejected, and that this affects the validity of her conclusions.

2.4.3 Grant and Baker’s Conclusions

Grant and Baker suggest that no trial is sufficient to demonstrate the validity of a marker as a general discriminant between authors, and that a theoretical understanding must be present as well.

They endorse Principal Component Analysis, the results of which cannot be generalised between different experiments.

2.4.4 Tests of potential author-identifying features: Short Substrings as Document Discriminators

This paper ([7]) tests the effectiveness of various kinds of short substrings in identifying documents. A nearest-centroid classifier see [8] was used to assign the documents to their correct classifications.

The first step here, as with Chaski, was to compile a database. The Bristol Benchmark Suite is the name of the compiled set of test problems.

Features tested:

1. counting frequencies of the 36 alphanumeric characters
2. counting frequencies of the most common 36 characters
3. counting frequencies of bigrams of characters
4. counting frequencies of trigrams of characters
5. counting frequencies of tetragrams of characters
6. counting frequencies of the most common 36 words
7. frequencies of doublets produced by Progressive Pairwise Chunking:
   most common pair of symbols found and replaced by code: repeat.
   the eventual substrings produced are known as doublets
8. frequencies of strings produced by Text Extending Feature Finding variation on Monte Carlo Feature Finding

extends the strings found by MCFF to the maximum possible length.

Chi-squared used to measure distinctiveness of random strings: the most distinctive 36 were kept.

9. frequencies of strings produced by Text Extending Feature Finding, square root count

distinctiveness is measured by chi-squared applied over the accumulated square root of the account produced by each kilobyte line

The features found to be most effective as document discriminators were, in order, 9, 8, and 2. The differences in the effectiveness of these methods was not significant: any of them could emerge as being the best, but none of the others could.

2.5 The Cusum Technique

The most frequently cited authority on this subject is [9].

2.5.1 An example Cusum Analysis

Farringdon provides an example cusum analysis in [10], where she analyses the following fragment of writing by DH Lawrence:

"I have a request to make. Perhaps you know that the [Name] asks for three Christmas stories, and offers a prize of three pounds for each. I have written two just for fun, and because [name] and [name] asked me why I didn’t, and so put upon me doing it to show I could. I may write a third. But one person may not send in more than one story. So will you send in the [kind] in your name? That is rather a sneeze, but I don’t see that it matters, for I make the story your property, and you will write it out again according to your tastewill you? It is the sort I want you to send, because it is the only one that is cast in its final form. I want you to write it out again in your style because mine would be recognised. Indeed you may treat it just as you like."
All the graphs presented here are from [10]
The basis of the cusum method is as follows:

- For each sentence in the sequence, the difference between its actual length and the mean length of the whole sequence is calculated. This set of differences is plotted cumulatively on a graph. (see Figure 2.1)

- The number of words displaying a certain habit (usually two or three letter words, or words that start with a vowel) in each sentence is also plotted cumulatively. Farringdon says: "Cusum analysts have found that there are nine tests which can be tested on samples. The three most common are the use of the 2 and 3 letter words (23lw); words starting with a vowel (initial vowel words, ivw); and the third is the combination of these two together (23lw+ivw), this last having often proved the most useful identifier of consistency. The other tests involve the use of words of four letters as well. One of these nine tests and sometimes more than one will prove consistent for a writer or speaker." (see Figure 2.2)

- The two graphs are then scaled proportionately to each other. (see Figure 2.3)

- The resulting lines overlap and look almost the same if the sequence of sentences is by the same author.

- If the sentences in the sequence are not by one author, the graph will show this– the lines will diverge or cross each other. (see Figure 2.4), which is an analysis of a mixture of writing by DH Lawrence and Aldous Huxley, where the writing by Aldous Huxley has been inserted at sentence 20.

Farringdon stresses the importance of not including atypically long or short sentences, from the sample, or sentences from the beginning or end part of texts. She cites success passing blind tests, and in using the technique in courts, and in cases of authorship attribution; her "test case" concerns the attribution of a short story believed to be by DH Lawrence.
Figure 2.1: Example Cusum graph
Figure 2.2: Example Cusum graph

The result of the habit of using words of 2 or 3 letters & words starting with a vowel in the same ten sentences.
Figure 2.3: Example Cusum graph

Two letters by D.H.Lawrence
a) written 1913 (20 sentences)
b) written 1907 (19 sentences)
Figure 2.4: Example Cusum graph
2.6 A Criticism of the Cusum Technique

De Haan, in [11], his review of [9] is fiercely critical of the book, and of the cusum technique as a whole. He points out several major inconsistencies in the content and tone of the book, and points out that "the cusum analyst decides what is typical for the user; in other words, 'anomalous' sentences are weeded out". It is hard not to concur with this statement, as the texts must be edited to a certain degree before they are analysed.

His major objection to the technique is that it is not linguistically motivated. Farringdon justifies the use of two to three letter words by pointing out that many function words have this number of letters; but of course many do not. He also makes the point that it is hard to reconstruct the kind of graphs that are reproduced in the book.

2.7 Neural Nets

A neural network is a software simulation of a "brain". The most important aspect of a neural network is that it can learn. A neural network is essentially a black box: it receives input and produces output but it is impossible to know what rules it is applying. The knowledge in a neural network is encoded in the weights attached to the links connecting the nodes of the network. For the weight connecting two nodes to change and strengthen, both nodes must have positive output simultaneously; in other words link strengthening is basically a function of local node activity. (see [12]) A Neural net has $n$ input units, a "hidden layer", and a predetermined number of outputs.

2.7.1 Neural Nets Applied to the Work of Shakespeare and Marlowe

Merriam and Matthews, in [12], create a neural net to discriminate between the works of Shakespeare and Marlowe. The discriminants chosen were the following five ratios:

- $\text{no}/T10$
- $(\text{of } x \text{ and})/\text{of}$
- $\text{so}/T10$
CHAPTER 2. REVIEW OF THE LITERATURE

- (the $x$ and)/ the
- with/T10

where $x$ is any word, and T10 is one of: but, by, for, no, not, so, that, the, to, with.

These discriminants were the inputs to a neural net.

A training corpus was selected, consisting of fifty 1000-word samples of Shakespeare’s work, and fifty 1000-word samples of Marlowe’s work.

The biggest problem for neural networks is overtraining, when the neural net learns to classify the training set perfectly, but loses the power to generalise when exposed to new data. To combat this problem part of the training set is held in reserve, so the success of the network is measured by its accuracy in classifying the never before seen part of the training set. This process is known as cross-validation.

The number of hidden units in the neural net is determined by the training process: too few leads to inaccuracy, and too many leads to a failure to generalise. In this case it was found that three was the right number of hidden units. Hence, the topography of this neural net is: five inputs (the discriminants above), three hidden units, and two outputs (Shakespeare or Marlowe).

The authors of the study found that the network was very successful in classifying the works of Shakespeare and Marlowe correctly, and that the instances where it failed were often instances where the authorship was in doubt in any case. Of anonymous plays, the network classified Edward III as Shakespeare, while The Contention and The True Tragedy were determined to be by Marlowe.

The authors additionally conclude that neural networks are a useful, promising tool for this purpose and hope they will be used more.

2.7.2 Bigram Frequencies and Neural Networks applied to the Federalist Papers

The Federalist Papers are each written by either Hamilton or Madison. The authorship of most is known; some are disputed. See [13] for more information.

In this case, the network was trained on all 14 papers known to be by Madison, and on 14 of the papers known to be by Hamilton.
The bigram frequency features chosen for input to the neural net were the $N$ bigrams with the largest difference in relative frequency across the two sets of papers. They were normalised before input to the net, as the very small frequency values involved would make for slow computation.

Part of what was tested in this study was the optimum value of $N$ the number of input features. It was found that results using 48 input bigram frequency features were good, but smaller numbers of features resulted in less accuracy.

Several configurations of neural nets were found that gave 100% accuracy in the training and testing sets. However these neural nets disagreed on the classification of the disputed papers. Since "unlike statistical methods, neural nets give no measure of confidence in their output", it is impossible to interpret these conflicting results.

The authors of the study conclude that neural nets are still useful as a "rough-and-ready" test of authorship, and say that taken as a group, the neural nets classified the papers as being by Hamilton, (which is generally believed by scholars to be the case.)
Chapter 3

Methods for Further Investigation
3.1 Methods Chosen

A huge array of potential author identification techniques were found on the review of the literature, far more than could possibly be investigated within the scope of this project. Probably the most interesting information was found in [2] and [7]. Both these papers detailed very simple, yet apparently effective, ways of discriminating between documents and authors.

3.1.1 Tags

Chaski’s most successful methods could not be emulated directly, as they would involve creating complex parsing software. However, variations on the theme can be created easily. Syntactic structures might be difficult to extract from an input corpus, however sequences of tags are not. There is tagging software readily available, (see Chapter 6 for more information.) Sequences of tags contain a great deal of the linguistically grounded syntactic information Chaski regards as so essential for successful author identification.

3.1.2 Characters

The information contained in [7] is also very intriguing. Can it really be the case that unigrams of characters are one of the most effective discriminators of documents? It is certainly an easy method to test.

3.1.3 Words

Word frequency is the most frequently used method historically and appears very often in the literature. It is also the logical extension of using characters and tags as discriminators, to use words as well.

3.2 Methods Not Chosen

3.2.1 Cusum Technique

The cusum technique ruled itself out as it is very labour intensive, and requires hand editing of the text before it is processed, and the graphs must be compared manually once produced. It was decided that a method which allows fully automated processing was a better idea.
3.2.2 Neural Nets

The difficulty with neural nets is that they allow no analysis of the results. If unsatisfactory results are produced, one has no idea why. If satisfactory results are produced, one still has no idea why. They may be useful for “rough-and-ready” classification purposes but they hold little interest in the context of an extensive investigation.

3.2.3 Others

Since there are hundreds of possible discriminants that could be used, most offering no clear advantage over any of the others, it was deemed a good idea to narrow the focus to the three broad categories given above, and the intuitively obvious variations that can be obtained by concordancing them. It is possible to drive the investigation a long way, looking at only what is listed in section 3.1.
Chapter 4

Algorithms and Mathematical Methods
4.1 Concordancing Algorithm

4.1.1 Introduction to Concordancers

The function of a concordancer is to break the input corpus down into
n-grams, where an n-gram is a group of n successive units of the input corpus.

An example will serve to make this clearer.

The quick brown fox ran.

When this sentence is broken down into word bigrams (an n-gram s.t. n=2), the bigrams are:

The quick
quick brown
brown fox
fox ran

When it is broken down into word unigrams (n=2), the unigrams are:

The
quick
brown
fox
ran

When it is broken down into letter bigrams, the bigrams are:

T h
h e
e SPACE
SPACE q
etc...

The concordancer performs these breakdowns.
4.1.2 Algorithm

All the concordancers work in the same way:

- When concordancing corpus into n-grams:
- Read corpus into an array $a$ (of letters, words, or tags)
- looping on $i$: for the $i$th item in $a$ {
  - make a string $s$ containing $a[i]...a[i+n-1]$]
  - use a hash to add 1 to the recorded frequency of $s$
- }

* See [14] if more information is needed on hashes, or on using them to record frequencies.

4.2 Chi Square Method for Calculating Significance

The chi square test is an algorithm for calculating the significance, or otherwise, of a statistical result.

4.2.1 An explanation of statistical significance and related terms


The Null Hypothesis

The first step in establishing whether a statistical pattern exists, is to make the assumption that it does not. This assumption is known as the Null Hypothesis.

Type 1 statistical errors

The Type 1 error in statistics is to derive the conclusion that a pattern exists—ie, that the null hypothesis does not hold—when in fact there is insufficient evidence to prove this.
CHAPTER 4. ALGORITHMS AND MATHEMATICAL METHODS

<table>
<thead>
<tr>
<th>ngram</th>
<th>Corpus1</th>
<th>Corpus2</th>
</tr>
</thead>
<tbody>
<tr>
<td>O of e</td>
<td>502</td>
<td>703</td>
</tr>
<tr>
<td>O of not-e</td>
<td>4046</td>
<td>5589</td>
</tr>
<tr>
<td>total ngrams</td>
<td>4548</td>
<td>6292</td>
</tr>
</tbody>
</table>

Figure 4.1: Example of a table for calculating chi square values

**Statistical significance**

This is a measure of the extent to which the results of the experiment deviate from the null hypothesis that "no pattern exists". This is the value which is calculated by the chi square algorithm.

**Significance threshold**

In order to avoid the type 1 error, it is usual for the experimenter to set a significance threshold prior to carrying out the experiment. If the significance value of the result is greater than the significance threshold, the result is deemed to be significant. In linguistics experiments the significance value chosen is often 0.05—ie if there is less than a 5% chance of the results occurring randomly, then the results are significant.

### 4.2.2 Why the chi square method?

Although other methods for calculating significance, such as the t-test, were examined (see [16] for more details), many of these tests have the important flaw that they assume a parametric distribution of experiment results. This is manifestly unsuitable for these purposes, as the frequencies of ngrams will in no way be parametrically distributed. The chi squared test is the most commonly used test of statistical significance in linguistics, and is versatile enough to be used here in many different contexts. There was no difficulty in finding ways to make it apply here.

### 4.2.3 The chi square calculation

The information here is culled from [16] and [17].
Steps in the chi square calculation

- Arrange the data in a table like that in Figure 4.1
- Calculate the expected frequency $E$ for each cell in the table. This is done by multiplying the (total observed frequency for the row), by (the total observed frequency for the column), and dividing the result by the total number of observations. The expected frequencies for Figure 4.1 are shown in Figure 4.2
- The formula for chi square is \((O - E)^2/E\) where $O$ is the observed frequency for the cell and $E$ is the expected frequency for the cell. Carry out this calculation for each cell in the table. The results of this calculation are shown in Figure 4.3.
- The total of the chi square value for each cell in the table is the chi square value for the table $c$. The result of this calculation is also shown in Figure 4.3.
- Find the degrees of freedom of the table. This is the (number of rows)-1 multiplied by the (number of columns)-1. In this case the number of degrees of freedom is equal to \((2 - 1)(2 - 1) = 1\).
- Look up the $c$ against the number of degrees of freedom, on a table of chi square values, to determine if the result is significant. If the significance threshold previously chosen is 0.05, then by reading the table of chi square values we can see that for the result to be considered significant it would have to be greater than 3.841. Hence, we can conclude that the distribution of the letter e does not vary significantly across these two corpora.

4.3 Algorithm for calculating which Ngrams Vary Significantly Across A Pair of Corpora, Using Chi Square

- Read in total frequency of corpus 1 into variable $f1$
- Read observed frequencies of corpus 1 into hash $h1$
### CHAPTER 4. ALGORITHMS AND MATHEMATICAL METHODS

#### Figure 4.2: Example of a table for calculating chi square values, with expected frequencies

<table>
<thead>
<tr>
<th>ngram</th>
<th>Corpus1</th>
<th>Corpus2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$ of e</td>
<td>502</td>
<td>703</td>
</tr>
<tr>
<td>$E$ of e</td>
<td>505.56</td>
<td>699.43</td>
</tr>
<tr>
<td>$O$ of not-e</td>
<td>4046</td>
<td>5589</td>
</tr>
<tr>
<td>$E$ of not-e</td>
<td>4042.3</td>
<td>4967.71</td>
</tr>
<tr>
<td>total ngrams</td>
<td>4548</td>
<td>6292</td>
</tr>
</tbody>
</table>

Chi square value for table: $(c) = 0.048$

#### Figure 4.3: Example of a table for calculating chi square values, with expected frequencies and chi square values

<table>
<thead>
<tr>
<th>ngram</th>
<th>Corpus1</th>
<th>Corpus2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$ of e</td>
<td>502</td>
<td>703</td>
</tr>
<tr>
<td>$E$ of e</td>
<td>505.56</td>
<td>699.43</td>
</tr>
<tr>
<td>chisquare value</td>
<td>0.025</td>
<td>0.018</td>
</tr>
<tr>
<td>$O$ of not-e</td>
<td>4046</td>
<td>5589</td>
</tr>
<tr>
<td>$E$ of not-e</td>
<td>4042.3</td>
<td>5592.6</td>
</tr>
<tr>
<td>chisquare value</td>
<td>0.003</td>
<td>0.002</td>
</tr>
<tr>
<td>total ngrams</td>
<td>4548</td>
<td>6292</td>
</tr>
</tbody>
</table>
• Read in total frequency of corpus 2 variable $f_2$
• Read observed frequencies of corpus 2 into hash $h_2$
• Calculate the total number of observations $n$ by adding $f_1$ and $f_2$
• For each observed value $v_1$ in of item $i$ in hash $h_1$
  • Find the corresponding observed value $v_2$ of item $i$ in $h_2$,
  • Calculate the expected values using: $n$ (total number of observations); $f_1$, $f_2$ (column totals); $h_1(i)$, $h_2(i)$, $f_1 - v_1, f_2 - v_2$ (observed frequencies); and $v_1 + v_2$, $f_2 + f_1 - v_1 - v_2$ (row totals)
• Calculate the chisquare value for this table; assign it to item $i$ in $h_3$
• endloop }
• Sort $h_3$ in descending order and print it; if the item has a significance larger than 0.05 (chisquare value larger than 3.841) indicate this in the output.

4.3.1 Discussion of Algorithm

The most notable problem with the algorithm used here is that it is not symmetrical. Comparing corpus1 with corpus2 will not necessarily give the same results as comparing corpus2 with corpus1. This occurs due to the fact that if corpus2 contains an ngram that corpus1 does not, it will not be included in the calculation.

4.4 Chi by Degrees of Freedom

The main problem with the chi square test is that when the sample size is increased, the null hypothesis is more easily rejected. This was independently rediscovered when working on bigrams and trigrams and finding that very large numbers of them exhibited significant differences even in text samples written by the same author. Obviously there will be more bigrams than unigrams and more trigrams than bigrams— and for great numbers of these the null hypothesis was rejected. The solution proposed by Kilgarriff in
[18]– see also [16] for more information– is a measure called *chi by degrees of freedom*. The chi by degrees of freedom measure is based on both word frequency information and chi square measures. Using it, a measure based on more words should be directly comparable to a measure based on fewer words. With this measure, each chi square measure is divided by the number of degrees of freedom of the table– eg, if 500 words are compared with each other the chisquare value is divided by $500 - 1 = 499$.

This provides us with a measure for producing similarity scores for pairs of entire corpora, and not just for pairs of ngram frequencies within corpora. This is exactly what was needed in the attempt to automatically assign a corpus to an author, which was not possible with the kind of data generated by the chisquare algorithm given above. In [18], Kilgarrif uses the measure for comparing vocabulary data across broadsheet and tabloid newspapers. In this investigation, we attempt to use the same method as a way of determining the author of a text.

### 4.5 Algorithm for calculating Corpus Similarity Scores using Chi By Degrees Of Freedom

#### 4.5.1 Initial Algorithm

- Read in value $m$
- Read in total frequency of corpus 1 into variable $f_1$
- Read observed frequencies of corpus 1 into hash $h_1$
- Read in total frequency of corpus 2 variable $f_2$
- Read observed frequencies of corpus 2 into hash $h_2$
- Calculate the total number of observations $n$ by adding $f_1$ and $f_2$
- Sort $h_1$ in descending order of observed frequencies
- For each of the highest $m$ observed value $v_1$ in of item $i$ in hash $h_1$
  - Find the corresponding observed value $v_2$ of item $i$ in $h_2$, 

• Calculate the expected values using: \( n, (\text{total number of observations}); \ f_1, f_2 (\text{column totals}); h_1\{i\}, h_2\{i\}, f_1-v_1, f_2-v_2 (\text{observed frequencies}); \text{ and } v_1+v_2, f_2+f_1-v_1-v_2 (\text{row totals}) \)

• Calculate the chisquare \( cs \) value for this table; print it to a file along with expected and observed values.

• add \( cs \) to a running total \( t \) of chisquare values

• endloop }

• Divide \( t \) by the number of degrees of freedom to give similarity score \( ss \)

• output \( ss \)

4.5.2 Further Automation of Algorithm

At first it was thought sufficient that this algorithm should calculate a similarity score for a pair of corpora. However, it was soon realised that this was insufficient in itself, as what we need is an algorithm that will compare every item in a list of corpora to every other item in the list, and generate a similarity score for each pair. When assigning a corpus to an author, we may comparing it to dozens of other corpora, and it would quickly become tiresome to run the same program over and over again; also a human would undoubtedly make mistakes and leave out some of the corpus pairs. So an extra step of automation was needed, and that extra step is included in the algorithm shown below.

• Read in list of corpora; assign to array \( a \)

• Read in value \( m \)

• Loop on \( j =1 \) to number of corpora{

• Loop on \( k= 1 \) to number of corpora{

• Read in total frequency of corpus \( j \) into variable \( f_1 \)

• Read observed frequencies of corpus \( k \) into hash \( h_1 \)

• Read in total frequency of corpus 2 variable \( f_2 \)
• Read observed frequencies of corpus 2 into hash $h_2$
• Calculate the total number of observations $n$ by adding $f_1$ and $f_2$
• Sort $h_1$ in descending order of observed frequencies
• For each of the highest $m$ observed value $v_1$ in of item $i$ in hash $h_1$
  • Find the corresponding observed value $v_2$ of item $i$ in $h_2$,
  • Calculate the expected values using: $n$, (total number of observations); $f_1, f_2$ (column totals); $h_1\{i\}, h_2\{i\}$, $f_1 - v_1, f_2 - v_2$ (observed frequencies); and $v_1 + v_2$, $f_2 + f_1 - v_1 - v_2$ (row totals)
  • Calculate the chisquare $cs$ value for this table; print it to a file along with expected and observed values.
• add $cs$ to a running total $t$ of chisquare values
• endloop }
• Divide $t$ by the number of degrees of freedom to give similarity score $ss$ for corpora $j$ and $k$
• output $ss$ to a file, along with the names of corpora $j$ and $k$
• endloop } for $k$
• endloop } for $j$

4.5.3 Discussion of Algorithm

This algorithm, while effective, is not symmetrical. It will produce two different similarity scores for each pair of corpora, depending on which corpus is $j$ and which corpus is $k$. There are two different reasons why this happens: firstly, we are comparing frequencies the most common $m$ ngrams in corpus $j$, and which $m$ ngrams are the most common is different for each corpus. Secondly, however, even when no $m$ value is assigned, and all the ngram frequencies in corpus $j$ are compared, the algorithm is still asymmetrical. This is because there are invariably ngrams present in one corpus that are not present in the other, even at large corpus sizes.
It is nonetheless very easy to generate a definitive similarity score for any given corpus pair by averaging the two values produced. However, it was unclear whether a truly symmetrical algorithm would be more effective, or, by averaging out differences between the two corpora, actually prove less effective. So it was decided to include this in the investigation.

4.6 The Symmetrical Algorithm for calculating Corpus Similarity Scores using Chi By Degrees of Freedom

- Read in list of corpora; assign to array \( a \)
- Read in value \( m \)
- Loop on \( j = 1 \) to number of corpora{
  - Loop on \( k = j + 1 \) to number of corpora{
    - Read in total frequency of corpus \( j \) into variable \( f_1 \)
    - Read observed frequencies of corpus \( k \) into hash \( h_1 \)
    - Read in total frequency of corpus \( 2 \) variable \( f_2 \)
    - Read observed frequencies of corpus \( 2 \) into hash \( h_2 \)
    - Calculate the total number of observations \( n \) by adding \( f_1 \) and \( f_2 \)
    - Find all ngrams that exist only in \( h_2 \); put these in hash \( h_3^* \)
    - Calculate the relative frequencies across both corpora of all items in \( h_1 \) and \( h_2 \); store this information in hash \( h_4 \)
    - Sort \( h_4 \) in descending order of relative frequencies
    - For each of the highest \( m \) relative frequencies in of item \( i \) in \( h_4 \{
      - Find observed value \( v_1 \) in of item \( i \) in hash \( h_1 \); if \( i \) does not exist in \( h_1 \), \( v_1 = 0 \)
• Find the corresponding observed value $v_2$ of item $i$ in $h_2$; if $i$ does not
exist in $h_2$, $v_2=0$
• Calculate the expected values using: $n$, (total number of observations);
$f_1, f_2$ (column totals); $h_1\{i\}, h_2\{i\}, f_1-v_1, f_2-v_2$ (observed frequen-
cies); and $v_1+v_2, f_2+f_1-v_1-v_2$ (row totals)
• Calculate the chisquare $cs$ value for this table; print it to a file along
with expected and observed values.
• add $cs$ to a running total $t$ of chisquare values
• endloop }
• Divide $t$ by the number of degrees of freedom to give similarity score
$ss$ for corpora $j$ and $k$
• output $ss$ to a file, along with the names of corpora $j$ and $k$
• endloop } for $k$
• endloop } for $j$

*for more information on this step, see [14]

4.7 The ”Ratio” Method for determining which
Program Specifications are most likely to
lead to successful Author Identification

4.8 Introduction

The end result of all the processing described in previous parts of this chapter,
is a list of corpus pairs, complete with similarity scores for those corpus pairs.
Looking at these files can give a rough idea of whether or not the configuration
tested is performing the function of author identification. However, in order
to systematise this kind of at-a-glance judgement, it was necessary to invent a
mathematical method for determining how effective any given configuration
was. The method that was invented is to take the average of the similarity
scores for all the pairs of corpora written by the same author, and the average of the similarity scores for all the pairs corpora written by different authors. The larger the ratio between these two values, the more effective the method was judged to be. The integrity of this method is somewhat in question: it would be better to find a proven statistical method that can calculate how closely the results come to the ideal ”bipolar” distribution. The particular problem is that it would be very easy for a configuration to classify some corpora wrongly, and still have a large ”ratio” measure thanks to extreme scores on correctly classified results. In other words this calculation method does not impose sufficient penalties on ”mistakes” made by a configuration. However, in the absence of a better way of measuring effectiveness, the ”ratio” method is a useful way of looking at the results of this experiment.

4.9 Algorithm for the Ratio Method

This algorithm will calculate the ratios of the results for many different experiments.

- Read corpus list into hash $h1$; assign the same value to corpora written by the same author.
  - For each results file $rf$ {
    - For each result in $rf$ {
      - If the corpora are by the same author according to $h1$ {
        - Make this similarity score part of the running average $a1$
      } else: corpora are by different authors{
        - Make this similarity score part of the running average $a2$
      }
      - endif
    } end loop: for each result }
  - Calculate and print the ratio $a1 : a2$ and the name of $rf$
  - Clear the variables $a1,a2$
  - end loop: foreach results file }
Chapter 5

The Concordancers
CHAPTER 5. THE CONCORDANCERS

5.1 Introduction

All the concordancers will normally be called by interface according to the rules described in 7.3 and Figure 7.1. This information is provided for purposes of program development and maintenance than for the end user. All concordancers operate according to the algorithm in 4.1.

5.2 concordance

This is a basic word concordancer.

5.2.1 Command Line Options in concordance

-n
Set to 1 for unigrams, 2 for bigrams, 3 for trigrams etc.

-i
The name of the input corpus.

-o
The name of the output file.

5.3 punconcord

This word concordancer counts items of punctuation as a word.

5.3.1 Command Line Options in punconcord

-n
Set to 1 for unigrams, 2 for bigrams, 3 for trigrams etc.

-i
The name of the input corpus.
-o
The name of the output file.

5.4 letter
This is a letter based concordancer, with some command line options to modify its behaviour.

5.4.1 Command Line Options in letter
-n
Set to 1 for unigrams, 2 for bigrams, 3 for trigrams etc.

-i
The name of the input corpus.

-o
The name of the output file.

-s
Binary switch. When it is set whitespace is counted as a letter.

-v
Binary switch. When it is set only consonants are counted.

5.5 tagconc.pl
This is a tag based concordancer, designed to process the output provided by the tagger at [19].
CHAPTER 5. THE CONCORDANCERS

5.5.1 Command Line Options in tagconc.pl

-n
Set to 1 for unigrams, 2 for bigrams, 3 for trigrams etc.

-i
The name of the input tagged corpus.

-o
The name of the output file.

5.6 keywords

This is not a concordancer, although it performs the same function in the program suite. It counts frequencies of key words, which are provided as input in a keyword file.

5.6.1 Command Line Options in keywords

-k
The name of the keyword file. The file should contain one word on each line, and no carriage return on the last line.

-i
The name of the input corpus.

-o
The name of the output file.
Chapter 6

About the Tagger
6.1 The Tagger

The tagger used is the Tree Tagger developed by the IMS at the University of Stuttgart. It is described in the papers at [19]. It uses the Penn-Treebank tagset, a description of which is also available at [19]

6.2 postprocess.pl

It was discovered that there is a problem with the tagger, which cannot handle the characters <, or >, and when it encounters these characters it outputs a line without any tags on it, interfering seriously with attempts to farther process the file. For this reason the program postprocess.pl was developed, which takes the output from the tagger and adds a new tag, HTML, to the Penn Treebank tagset, to cover instances of these characters occurring. It should be noted that although these characters are most likely to occur in the processing of HTML files, they are also used frequently in email by some people, as in the symbol <g> for ”grin”.

6.3 Relevant filenames

When interface is called with a -t tag, it calls the tagger which outputs the tagged corpus to corpusname_tagged.temp. postprocess.pl then takes corpusname_tagged.temp and outputs the results of its processing to corpusname_tagged.
Chapter 7

How To Use the Programs
7.1 chisquare

This calculates which ngrams can be said to vary significantly in terms of the relevant frequency across two corpora. Significance is calculated using the chi-square method, as detailed in 4.2.

Usage of program:

```
chisquare corpusname1 corpusname2 optionalnumber
```

The expected format of the corpus files is that

- the total number of ngrams in the corpus will be the first line of the files corpusname1 and corpusname2.
- Record separator= return (standard line break)
- Field separator= whitespace
- the labels of the ngrams will be in the first field.
- the frequency count for ngrams is the second field.

The last two assumptions are not always accurate. For example, with letter bigrams, output from the letter concordancer will have the format

```
example line:
a b 35
```

for the case where the letter bigram ab has a count of 35. In this case the frequency is in the third column. That is the reason for the optional numerical argument on the command line. Files that have this format can be processed using the command line input

```
chisquare corpusname1 corpusname2 3
```

where 3 is the field number in which the frequency appears. When counting trigrams using output from the concordancers provided here the command line should be

```
chisquare corpusname1 corpusname2 4
```

as these files will have 3 letters separated by whitespace on each line, and then the frequency appears in the fourth field. The contents of all fields before the field designated as containing frequency are used as a label for the ngram in question.

This program produces output such as that shown in 8.1.2.
7.2 multicdf

The function of this program is as follows

- Input multiple concordanced files corresponding to the corpora being compared
- Use the Chi by Degrees of Freedom method detailed in 4.4 to produce a value for similarity of each pair of corpora input.
- output a ranked table showing the similarity score for each pair of corpora on the list (strictly, for each concordanced file on the list: it is up to the human or program providing input for this program to make sure the comparisons executed by multicdf are sensible comparisons between corpora)
- output numerical tables detailing the calculations performed to in the process of obtaining similarity scores for corpus pairs
- do all this in a variety of flexible ways depending on command line options specified by the user of the program.

See 4.5.2 for the algorithm used by this program.

7.2.1 Command line options for multicdf

-f

The -f command line option is used as follows

    -f filename

where the file filename contains the list of concordanced files to be processed.

It is assumed that filename consists solely of a list of filenames separated by carriage returns; blank lines or a carriage return at the end of the file may lead to program errors.

This option is compulsory and the program will not run if it is not set.
-m

As described in 4.4 this method takes the frequencies of the top m most frequent n-grams in both files and compares them. This option is used as in the following example:

\[-m\ 40\]

will set the value of m to be 40, so that the most common 40 n-grams in the first file are used in the calculation of the similarity value.

The m value defaults to zero, if the -m option is not used, which the program interprets to mean every n-gram in the first file should be used in the calculation of the similarity value.

-b

The numerical tables are very long and bulky files, particularly when using bigrams or trigrams with the value of m set to zero. The default is to output these files to the tmp directory, as should the user exceed their disk quota no farther data will be saved and the potential for loss of work is considerable.

However, if disk space is not an issue, or unigrams or small values of m are being used, the user may prefer that the .tables files are saved somewhere permanent. In this case add

\[-b\]

to the command line and these files will be written to the local directory.

-n

If processing unigrams, set

\[-n\ 1\]

If processing bigrams set

\[-n\ 2\]

If processing trigrams set

\[-n\ 3\]

n defaults to two.
The name of the output file is supplied using the -o option. Usage:

```
-o outputfile
```

The longer numerical tables detailing the calculation are always output to `outputfile.tables` -o defaults to `cbdfout` if left undefined, meaning that the default file in this case for outputting the numerical tables to is `cbdfout.tables`.

### 7.2.2 General assumptions made by `multicdf`

It is assumed that the files used, and referenced in `-f filename` are generated by one of the concordancers that were programmed for this purpose, ie that they have the following qualities:

- the total number of ngrams in the corpus will be the first line of the concordanced files.
- Record separator= return (standard line break)
- Field separator=whitespace
- the labels of the ngrams will occupy the first n fields.
- the frequency count for ngrams is found in the n+1th field.

It is assumed that only like is compared with like. For example, it would be a nonsense to compare files concordanced for word bigrams with a file concordanced for letter bigrams, or word unigrams, and any output produced from so doing will be meaningless.

### 7.2.3 Sample Output And Input

The program `multicdf` is asymmetrical. That is, the similarity value calculated between corpus 1 and corpus 2 is not the same as the similarity value calculated between corpus 2 and corpus 1. This occurs because, as described in 4.6 the first m ngrams in corpus 1 will not be the same as the first m ngrams in corpus 2, so different values are used in the calculation. This program performs the same functions as `multicdf` but it calculates what the
m most common ngrams across both corpora are, based on the sum of the relative frequencies in each. See 4.6 for the algorithm used by this program.

It will be noted that since this program is symmetrical in its output, it outputs fewer lines in its results.

7.2.4 Command Line Options for multicdf.new

These are identical to what is described in 7.2.1.

7.2.5 Assumptions made by multicdf.new

See 7.2.2.

7.3 interface

The function of this program is as follows:

- Input a list of corpora to be concordanced.
- Concordance the files by running the requisite concordancer, with the necessary pre-processing in the case of the files in which tags are to be concordanced. Output the resulting files to disk.
- Calculate the similarity scores on the newly concordanced files by running multicdf or multicdf.new (the cbdf=Chi By Degrees of Freedom programs) as specified. Output the resulting ranked table of similarity scores and the larger tables of calculations to disk.
- Vary the parameters by which similarity scores are calculated according to the users’ specifications on the command line.
- Implement a systematic naming schema for files generated by this program and by the programs called by this program.

The subprograms called by this program, and the command line options that call them, are shown in Figure 7.1

The value of a systematic naming schema should not be underestimated in this context. Hundreds of files have been generated in the course of this
investigation and it would have been impossible to keep track of them without the names generated by `interface`.

It is highly recommended that users of this system should generate all similarity score tables by calling on `interface` and not by calling on concordancer and then `multicdf` directly, in order to take advantage of this schema. Systematic filenames are also essential if farther processing is to be done on the similarity score tables.

### 7.3.1 Command line options for `interface`

**-f**

The `-f` command line option is used as follows

```
-f filename
```

where the file `filename` contains the list of corpus files to be processed.

It is assumed that `filename` consists solely of a list of lines which have the filenames as their FIRST filed. Other fields are ignored. Blank lines or a carriage return at the end of the file may lead to program errors. FILENAMES SHOULD NOT CONTAIN PERIODS.

This option is compulsory and the program will not run if it is not set.

**-n**

If processing unigrams, set

```
-n 1
```

If processing bigrams set

```
-n 2
```

If processing trigrams set

```
-n 3
```

n defaults to two. This option is passed on to the concordancer and the cbdf programs.
Figure 7.1: flowchart for program interface
-m
As described in 4.4 this method takes the frequencies of the top m most frequent ngrams in both files and compares them. This option is used as in the following example:

    -m 40

will set the value of m to be 40, so that the most common 40 ngrams in the first file are used in the calculation of the similarity value.

The m value defaults to zero which means that in this case either all the ngrams in the first file will be used, or all the ngrams in both files will be used as a basis for calculation, depending on how the -x setting is set. (See 7.2.3 and 4.6)

-w
Binary switch.
Concordance words: ie use the program concordance or, if -z is set, punconc when concordancing. If the -c switch is not set, the program will look for the filenames which would have been outputted from the word based concordancers.

-l
Binary switch.
Concordance letters: ie use the program letter when concordancing. If the -c switch is not set, the program will look for the filenames which would have been outputted from the letter based concordancer.

-t
Binary switch.
Concordance tags: ie use the program tagconc when concordancing.
If the -t switch is set, the program assumes that the corpora have not already been tagged. It will then start to tag every corpus on the list, and call the program postprocess.pl to eliminate the problem with angled brackets (see 6.2. Tagging is a slow process. This is a thing to be careful of. The -c switch will presumably always be set when the -t switch is called. If the corpora have not been tagged, the tags cannot be already concordanced.
-p

Binary switch.

Concordance tags: ie use the program tagconc when concordancing.

If the -p switch is set, the program assumes that the corpora have already been tagged. It will then look for the files which would have been generated by postprocess.pl which will have filenames of the form corpus-name.tagged. If the tagged corpora are called something else they must be renamed in this format.

If the -c switch is not set, the program will look for the filenames which would have been outputted from the tag based concordancer.

-c

Binary switch.

The stated purpose of this program is simple: for any given list of corpora, to run a concordancer, and then run one of the cbdf programs on the results. However it is not always necessary to re-concordance the corpora when running experiments. If the intention is for example to vary the value of m, or to experiment with the -x switch, then re-concordancing for each new experiment is an unnecessary waste of time. Unless the -c switch is set, the concordancer will not run, and it will be assumed that the relevant automatically named output from the concordancer exists in the directory.

-b

Binary switch.

The numerical tables output from the cbdf programs are very long and bulky files, particularly when using bigrams or trigrams with the value of m set to zero. The default is to output these files to the tmp directory, as should the user exceed their disk quota no farther data will be saved and the potential for loss of work is considerable.

If the -b switch is set these tables will be written into the local directory. This switch is passed directly on to the cbdf programs.

-x

Binary switch.
CHAPTER 7. HOW TO USE THE PROGRAMS

If this switch is set, the symmetrical method of calculation is used, i.e. `multicdf.new` is called instead of `multicdf`. See 7.2.3 and 4.6.

-v
Binary switch.
This switch is valid only when combined with the -l switch. It only counts the consonants. It is passed directly on to the program `letter`.

-s
Binary switch.
This switch is valid only when combined with the -l switch. Whitespace is counted as a letter, although multiple whitespace is not counted multiple times. It can be combined with the -v switch.

-k
Not a binary switch. Input is

-k `keywordfile`

Concordancing is done on the basis of the keywords included in `keywordfile`. It is assumed that `keywordfile` consists solely of a list of keywords separated by carriage returns, with no blank lines. No other switches apart from -l operate at the same time as the -k switch.

-z
Binary switch.
Valid only when combined with the -l switch. The -z switch includes punctuation in the word concordancing, by calling `puncord` instead of `concordance`. The default with -w is not to include punctuation.

7.3.2 General assumptions made by interface

Aside from the assumptions about input files mentioned in the previous section, it is assumed that one of -p,-t,-l, -k or -w will be set to true. If none of these is defined, results may be unexpected.
7.4 \textbf{mean}

The aim of this program is to process the output tables generated by the cbdf programs. As described in 4.7 the main processing done is to calculate the mean similarity score of corpora written by the same author, and then the mean similarity score of corpora written by different authors, and then output the ratio between the two. (SD ratio) Corpora that are not assigned to any author in the input are then assigned to one of the authors listed in the corpus. See 4.7 and 4.9 for a discussion of this method and the algorithm used by this program.

7.4.1 \textbf{Command Line Switches in mean}

-\texttt{f}

usage:

\texttt{-f corpuslist}

\textit{corpuslist} should be a list of the corpus files arranged as in table 7.1: in the case where the first and fourth corpora are written by the same author, the second by a different author, and the author of the third corpus is to be identified. It is assumed that EITHER input lines consist of two fields, the first field containing the name of the corpus and the second field containing the name or some other unique identifier of the author, fields being separated by whitespace and lines by carriage returns OR input lines consist of corpus names only separated by carriage returns, and in this case there will be no attempt to assign corpora by "unknown" authors to an author in the group, and only the SD ratios will be calculated.
-r

usage:

    -f rankfilelist

    rankfilelist should consist of a list of output files from running interface. All the ranked tables of similarity scores produced by interface rm are prefixed by the name of the corpus list that was fed to it, and suffixed with .rank.

    so the input file here, rankfilelist can be generated using the command

        ls -1 corpuslist*.rank > rankfilelist

The -1 switch on /tt ls /rm ensures that only one filename is listed per line on the output.

-n

    -n n

For each corpus marked with a 0 in corpuslist the program calculates similarity scores for each set of corpora that are marked with the same author. The top n similarity scores are then output for each file in rankfilelist. The advantage of using this method, instead of taking the more intuitively obvious approach of printing out only which corpus set is the best match for which "unknown author" corpus, is that other data is revealed here about the reliability of the method being tested. Given two unknown author corpora, A and B, written by authors A and B respectively, the ideal method would have corpus A being assigned to author A, and corpus B being assigned to author B, as its highest probability matches. If the three most probable matches are

1. corpus A --> author A
2. corpus A --> author B
3. corpus B --> author B

then the method being tested is something less than the ideal, even though it does assign all the corpora to the correct author. This extra information
is useful to avoid relying on the SD ratio alone as a measure of the quality
of a method.

As a general rule for assigning n, if there are x unknown author corpora
in the corpuslist then a good value for n would be x+1.

7.4.2 Examples of Input and Output

7.5 An Example Program Run

This section deals with an example of using the program suite to determine
the best methods of author identification, or to identify the author of a
particular corpus.

7.5.1 Corpora

Begin with a directory containing the corpora you want to process.

sun21> ls
  tham1 tham2 zort1 zort2 zort3

In this case the corpora in question are tham1 and tham2, written by
the same author, and zort1, zort2, and zort3, all written by the same
author.

7.5.2 Corpus File

Let us assume that we are trying to identify the author of the corpus zort3.
In this case, of course, the author is actually known, so the program run
serves as a way of testing the best methods of author identification in this
case. However, in the case where we are trying to assign the author of an
actually unknown corpus, the procedure is exactly the same, except that in
that instance it is probably best to restrict our runs to a few methods already
proven to be successful (see Conclusions) whereas in a situation where we are
comparing methods it is possible to go wild using many unlikely combinations
of settings in interface. In this example instance we will restrict the
number of runs of interface to three, for the sake of clarity.

The corpus file should be in the same directory as the corpora: here the
corpus list is called corpora.
As is shown here, the corpus list has the filenames of the corpora being analysed in the first field of each line. In the second field of each line is a code, assigning each corpus to its author. It makes no difference what the code is, provided it is not 0 (numerical) and identical where the author is identical. As can also be seen from the terminal output above, the last line of the corpus file should NOT have a carriage return at the end. Any corpus where the author is to be determined by the program should be assigned the author code 0 (numerical).

7.5.3 Running interface

Example Run 1

```
sun21> /FYP/interface -f corpora -x -n 1 -l -c
```

The meaning of this command is: Run the program `interface` on the corpora listed in the file `corpora` (-f option) using the symmetrical method of calculation (-x option) using unigrams (-n option) of letters (-l option), and concordance the corpora first (-c option). The program outputs to the terminal some PERL warnings, which are entirely irrelevant and invalid and should be ignored, the list of the filenames as it processes them, and the ranked list of similarity scores (which is also output to a file with the suffix `.rank`).

Needless to say the directory path for `interface` should be replaced with the path extant on the system the user is currently running.

After this run the directory contains the following files and information:

```
sun21> ls
```
Several files have been generated by this run. Files ending in the suffix .l1 are files generated by the concordancer (in this case letter). These are the concordanced versions of whichever corpus is indicated by the first part of the filename— the suffix .l1 indicates the files have been concordanced for letter unigrams. The file corpora.l1.conclist contains a list of the filenames of files generated by the concordancer. As explained in 7.3, the program interface calls a concordancer, and then calls multicdf. corpora.l1.conclist is generated by interface as necessary input for multicdf. The file corpora.l1.m0.x.rank is a list of the results so far. All the pairs of corpora are given similarity scores based on the method of calculation specified when interface was called. They are arranged in reverse order of
similarity.

**Example Run 2**

Here we generate another run based on letter unigrams.

```bash
sun21> ~/FYP/interface -f corpora -x -n 1 -l -m 30
```

This run compares only the values of the thirty most common unigrams across both corpora (by setting the -m switch). Note that the -c switch is NOT set; it is unnecessary and a waste of CPU time, as these corpora are already concordanced for letter unigrams.

```bash
sun21> ls
corpora tham1.l1 zort2
corpora.l1.conclist tham2 zort2.l1
corpora.l1.m0.x.rank tham2.l1 zort3
corpora.l1.m30.x.rank zort1 zort3.l1
tham1 zort1.l1
```

Only one extra file is generated by this run: `corpora.l1.m30.x.rank` which has the same format as `corpora.l1.m0.x.rank`, the contents of which are shown above.

**Example Run 3**

Here we generate a run based on letter bigrams.

```bash
sun21> ~/FYP/interface -f corpora -x -n 2 -l -c -m 40
```

This generates similarity scores based on the top 40 (-m switch) letter (-l) bigrams(-n) across all the pairs of files in `corpora` (-f). Note that the -c switch must now be set, as these corpora have not been concordanced for letter bigrams.

Several new files are generated by this run:

```bash
sun21> ls
corpora tham1.l1 zort1.12
  corpora.l1.conclist tham1.12 zort2
  corpora.l1.m0.x.rank tham2 zort2.11
```
corpora.l1.m30.x.rank  tham2.l1  zort2.l2
corpora.l2.conclist  tham2.l2  zort3
corpora.l2.m40.x.rank  zort1  zort3.l1
tham1  zort1.l1  zort3.l2

All the concordanced files for letter bigrams have been generated (corpus-name.l2), plus the list of their names. (corpora.l2.conclist). Also, of course, the ranked list of similarity scores has been generated. (corpora.l2.m40.x.rank) And should more runs based on letter bigrams be desired, of course it is not necessary to set the -c switch.

7.5.4 Creating the list of .rank files

    sun21> ls -1 corpora*rank >ranklist
    sun21> cat ranklist
    corpora.l1.m0.x.rank
    corpora.l1.m30.x.rank
    corpora.l2.m40.x.rank

This list is needed by the program .mean in order to compare the similarity scores across the different methods. Use the -1(numeric) switch in order to ensure that only one filename per line is printed to the output file.

7.5.5 Running mean

    sun21> ~/FYP/mean -f corpora -r ranklist -n 2
    (warning messages) ...

    corpora.l1.m0.x.rank  Right: 2.4777138204839  Wrong: 10.5189829882362
    Ratio: 4.24543904193983
    zort3.z           zort3 fits in category z with score 2.21671866603487
    zort3.t           zort3 fits in category t with score 7.65715343226872
    corpora.l1.m30.x.rank  Right: 3.17180177002685  Wrong: 10.8645591863456
    Ratio: 3.42535882570418
    zort3.z           zort3 fits in category z with score 1.92315377513974
    zort3.t           zort3 fits in category t with score 5.76668713496033
    corpora.l2.m40.x.rank  Right: 2.01091558065793  Wrong: 4.82561295060875
This is the final output from the program run, and it is considerably simpler to interpret than it appears at first glance. The first thing we see is that all the methods we tested assigned the corpus \texttt{zort3} to its correct category. We can tell this because the program outputs the top n category/corpus combinations in order of preference. In each case the preferred category for corpus \texttt{zort3} is \texttt{z}. We can also tell that some methods were more definite about this category assignation than others. We can tell this by looking at the Ratio value. This is the ratio of (the mean similarity scores of assignations of the corpus to the wrong category) to (the mean similarity scores of assignations of the corpus to the right category). See 4.7. The higher the Ratio value, the more certain the method was in how it assigned the category... it can be taken as a rough measure of how ”good” the method is, in cases where more files are tested, and all of them were correctly assigned. In this case we would conclude that the letter unigrams with m value not assigned (\texttt{corpora.11.m0.x.rank}) is the best method out of the three we tested.

7.6 Deciphering The Filenames

7.6.1 Concordanced Files

```
sun21> ls tham1*
tham1    tham1.l3    tham1.t1    tham1.w!2
tham1.k_key    tham1.1s.2    tham1.t2    tham1.w1
tham1.k_keywords    tham1.1s1    tham1.t3    tham1.w2
tham1.l1    tham1.1s2    tham1.tagged    tham1.w3
tham1.l2    tham1.lv1    tham1.tagged.temp
tham1.lvs1    tham1.w!1
```

This is a listing from one of the experimental directories, of all the concordanced files generated during the experiment that are based on the corpus \texttt{tham1}.

The codes in the suffixes correspond to the command line switches that were set when the program \texttt{interface} was run.
CHAPTER 7. HOW TO USE THE PROGRAMS

tham1
The original corpus.

tham1.k_key
run using the -k keyword setting, and the keyword file key.

tham1.k_keywords
run using the -k keyword setting, and the keyword file keywords.

tham1.l1
run using the letter -l setting, and -n 1.

tham1.l2
run using the letter -l setting, and -n 2.

tham1.lvs1
run using the letter -l setting, and -n 1, with the -v and -s switches set (see 7.3.1)

tham1.ls2
run using the letter -l setting, and -n 2, with the -s switch set (see 7.3.1)

tham1.w1
run using the word -w setting, and -n 1.

tham1.w2
run using the word -w setting, and -n 2.
tham1.w!2
run using the word -w setting, and -n 2, with the -z switch (which counts punctuation as a word) set. The -z switch contributes the ! to the filename.

tham1.tagged.temp
This file is the tagged corpus, as output directly from the tree-tagger. It will have been generated when interface was run with the -t switch set. It can be deleted assuming the file tham1.tagged exists.

tham1.tagged
This file is the tagged corpus, as output from the program which postprocesses the tagger output. It will have been generated when interface was run with the -t switch set.

tham1.t1
run using the tag -p or -t setting, and -n 1.

tham1.t3
run using the tag -p or -t setting, and -n 3.

7.6.2 Lists of Concordanced Files
These files are prefixed with the name of the corpus list, and suffixed with .concllist.

sun21> ls lj*concllist
lj.k_key1.concllist  lj.ls2.concllist  lj.w!1.concllist
lj.k_keywords1.concllist lj.lv1.concllist  lj.w!2.concllist
lj.11.concllist  lj.lvs1.concllist  lj.w1.concllist
lj.12.concllist  lj.t1.concllist  lj.w2.concllist
lj.13.concllist  lj.t2.concllist  lj.w3.concllist
lj.ls1.concllist  lj.t3.concllist

sun21> cat lj.ls1.concllist
friends1.ls1
CHAPTER 7. HOW TO USE THE PROGRAMS

friends2.ls1
friends3.ls1
madp1.ls1
madp2.ls1
tham1.ls1
tham2.ls1
tham3.ls1
zort1.ls1
zort2.ls1
zort3.ls1

These are all the lists of concordanced files generated for the corpus list ly, in the same experimental directory. They follow the same naming patterns as the concordanced files themselves: each file in a list is generated by the same run of interface -c.

7.6.3 Results Files

These files are prefixed with the name of the corpus list, and suffixed with .rank.

sun21> ls lj*rank
lj.k_key1.m0.rank     lj.12.m50.rank     lj.w!1.m0.rank
lj.k_key1.m0.x.rank   lj.12.m50.x.rank   lj.w!1.m0.x.rank
lj.k_keywords1.m0.rank lj.13.m50.rank     lj.w!2.m40.rank
lj.11.m0.rank         lj.1s1.m50.x.rank   lj.w!2.m40.x.rank
lj.11.m0.x.rank       lj.1s2.m50.rank     lj.w1.m0.rank
lj.11.m100.x.rank     lj.lv1.m50.x.rank   lj.w1.m0.x.rank
lj.11.m40.x.rank      lj.lvs1.m50.x.rank   lj.w1.m10.x.rank
lj.11.m50.rank        lj.t1.m0.rank      lj.w1.m50.rank
lj.11.m50.x.rank      lj.t1.m0.x.rank     lj.w2.m0.rank
lj.11.m60.rank        lj.t1.m30.rank      lj.w2.m0.x.rank
lj.11.m60.x.rank      lj.t1.m30.x.rank     lj.w2.m100.x.rank
lj.12.m0.rank         lj.t2.m0.rank       lj.w2.m50.rank
lj.12.m0.x.rank       lj.t3.m0.rank       lj.w3.m50.rank

Some examples of how to read the filenames follow. They follow the pattern corpuslist.concordancesuffix.mvalue.(x).rank
CHAPTER 7. HOW TO USE THE PROGRAMS

lj.t2.m0.rank
• the corpus list used was lj
• the concordance suffix means tag bigrams (see 7.6.1)
• the m value was not set... all the bigrams in both corpora were used in the calculation. (m0)
• there is no .x so the symmetrical method of calculation was not used.

lj.ls2.m50.rank
• the corpus list used was lj
• the concordance suffix means letter bigrams, with the -s option set (see 7.6.1)
• the m value was set to 50... the most common 50 bigrams were used in the calculation (m50)
• there is no .x so the symmetrical method of calculation was not used.

lj.l1.m100.x.rank
• the corpus list used was lj
• the concordance suffix means letter unigrams (see 7.6.1)
• the m value was set to 100... the most common 100 bigrams were used in the calculation (m100)
• there is a .x so the symmetrical method of calculation was used.

lj.k_key1.m0.x.rank
• the corpus list used was lj
• the concordance suffix means keyword unigrams were used from the file key (see 7.6.1). There is no option to use bigrams or trigrams with keyword concordancing.
• the m value was not set... all the keywords in both corpora were used in the calculation. This would be the normal state of affairs in keyword concordancing

• there is a .x so the symmetrical method of calculation was used. In this case it makes no difference whether it is used or not, as it would be expected that all the keywords would appear in both files, and assuming m is set to 0 the results of symmetrical and asymmetrical calculation will be the same.
Chapter 8

Results and Discussion
8.1 Which Ngrams are likely to be significant?

8.1.1 Corpora

For this question a corpus was compiled from a mailing list, where all the posters were writing about the same topic, at a generally high level of discussion. It is reasonable to assume that the majority of the posters to this list are female and at least college educated. They are unlikely to make many spelling mistakes or use ”AOLSpeak” spelling. A corpus was compiled from all the posts made the list, and another corpus was compiled from posts made by Yahtzee, one of the most frequent posters to the list.

8.1.2 Which single letters were the most significantly different?

<table>
<thead>
<tr>
<th>Letter</th>
<th>Significance</th>
<th>File</th>
<th>Frequency</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>14.483</td>
<td>g1results</td>
<td>0.003</td>
<td>y1results</td>
</tr>
<tr>
<td>.</td>
<td>6.935</td>
<td>g1results</td>
<td>0.015</td>
<td>y1results</td>
</tr>
<tr>
<td>t</td>
<td>6.560</td>
<td>g1results</td>
<td>0.073</td>
<td>y1results</td>
</tr>
<tr>
<td>L</td>
<td>6.373</td>
<td>g1results</td>
<td>0.002</td>
<td>y1results</td>
</tr>
<tr>
<td>S</td>
<td>4.740</td>
<td>g1results</td>
<td>0.005</td>
<td>y1results</td>
</tr>
<tr>
<td>O</td>
<td>4.735</td>
<td>g1results</td>
<td>0.002</td>
<td>y1results</td>
</tr>
<tr>
<td>!</td>
<td>4.339</td>
<td>g1results</td>
<td>0.001</td>
<td>y1results</td>
</tr>
<tr>
<td>M</td>
<td>3.773</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>3.773</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3.494</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`y1results` is the file of letter unigrams for Yahtzee, `g1results` the letter unigrams for the general corpus. The numbers next to the filenames are the relative frequencies of the unigrams within the corpora. The unigrams are marked as significant if the significance of their variation across corpora is greater than 0.05. Surprisingly, a pattern can be discerned here: Yahtzee is clearly not given to using asterisks for emphasis, as some emailers are. She doesn’t use exclamation marks. The meaning of other results is more mysterious—why is Yahtzee so inexplicably fond of the letter t? And why use less full stops? Further examination of these corpora may provide the answers.
8.1.3 Which bigrams were the most significantly different?

```
perl chisquare2 c2results y2results 3 | head -n 20

. . 12.307 SIGNIFICANT c2results 0.003 y2results 0.000
o g 7.185 SIGNIFICANT c2results 0.000 y2results 0.002
c r 7.185 SIGNIFICANT c2results 0.001 y2results 0.003
d o 7.096 SIGNIFICANT c2results 0.004 y2results 0.001
m a 6.303 SIGNIFICANT c2results 0.005 y2results 0.002
r e 5.709 SIGNIFICANT c2results 0.012 y2results 0.018
t i 5.205 SIGNIFICANT c2results 0.006 y2results 0.010
l l 5.056 SIGNIFICANT c2results 0.010 y2results 0.006
a t 4.985 SIGNIFICANT c2results 0.009 y2results 0.013
m i 4.521 SIGNIFICANT c2results 0.001 y2results 0.003
da 4.521 SIGNIFICANT c2results 0.001 y2results 0.003
e k 4.339 SIGNIFICANT c2results 0.001 y2results 0.000
k e 4.263 SIGNIFICANT c2results 0.004 y2results 0.002
( a 4.221 SIGNIFICANT c2results 0.000 y2results 0.001
r o 3.994 SIGNIFICANT c2results 0.000 y2results 0.005
l u 3.976 SIGNIFICANT c2results 0.001 y2results 0.002
I s 3.616
```

One of the questions posed in the previous section now seems to be answered. Yahtzee uses less full stops because she uses less ellipses—no ellipses at all, it would appear. On the whole, though, these results raise more questions than they answer. It seems that Yahtzee might use the word "at" more frequently than average. But can the high occurrence of the combination og really be a mark of author identification? It seems on the surface more likely to be a topic related blip. Looking at the words may help.

8.1.4 Which words were the most significantly different?

```
perl chisquare g1words y1words | head

m e  5.847 SIGNIFICANT  g1words 0.006  y1words 0.000
a  4.635 SIGNIFICANT  g1words 0.016  y1words 0.030
no  4.382 SIGNIFICANT  g1words 0.004  y1words 0.000
plagiarism  4.186 SIGNIFICANT  g1words 0.001  y1words 0.005
```
CHAPTER 8. RESULTS AND DISCUSSION

The occurrence of "plagiarism" in this table comes about because that was the topic of discussion on the mailing list. None of the significantly-different words cast any light on the letter bigrams above.

\subsection{Which tags were the most significantly different?}

\verbatim
perl chisquare y1tags g1tags | head
NP 24.373 SIGNIFICANT y1tags 0.024 g1tags 0.065
DT 4.540 SIGNIFICANT y1tags 0.090 g1tags 0.068
WRB 3.849 SIGNIFICANT y1tags 0.002 g1tags 0.007
IN 2.777 y1tags 0.109 g1tags 0.090
SENT 2.661 y1tags 0.035 g1tags 0.048
\end{verbatim}

So we gather that Yahtzee uses less proper nouns, more determiners (is that why she uses more letter t?), fewer WH-adverbs and fewer sentence boundaries, but more prepositions. These all do sound like likely markers of author identification, if what "sounds likely" has any correlation at all with "what is a useful marker".

\section{Which tag bigrams were the most significantly different?}

\verbatim
perl chisquare y2tags g2tags 3 |head
NP NP 8.558 SIGNIFICANT y2tags 0.002 g2tags 0.011
SENT NP 8.522 SIGNIFICANT y2tags 0.001 g2tags 0.009
NP POS 4.960 SIGNIFICANT y2tags 0.001 g2tags 0.006
NP VBZ 4.264 SIGNIFICANT y2tags 0.001 g2tags 0.006
NN TO 4.222 SIGNIFICANT y2tags 0.007 g2tags 0.002
VBP VBG 4.173 SIGNIFICANT y2tags 0.004 g2tags 0.001
CC IN 4.115 SIGNIFICANT y2tags 0.003 g2tags 0.000
( CC 4.115 SIGNIFICANT y2tags 0.003 g2tags 0.000
) VB 4.115 SIGNIFICANT y2tags 0.003 g2tags 0.000
NP NN 3.576 y2tags 0.001 g2tags 0.005
\end{verbatim}

So we can see that Yahtzee does not go in for proper nouns at all. She does, however, like to use nouns followed by "to", and non third person verbs
followed by gerunds. It might be possible to consider that last in the light of
her non-usage of the word ”me” and wonder if she is more inclined to talk
about herself as ”I”– are all these non third person verbs first person ones?
However, there is insufficient evidence her to conclude that.

8.1.6 Is it possible to identify the author on the basis
of this kind of evidence?

The output above was based on an attempt to assign an email written
by Yahtzee to the correct corpus. The email was only 327 words long. The
letter unigrams and the tag unigrams succeeded in assigning the email to
the Yahtzee corpus rather than the general corpus. The letter bigrams, tag
bigrams and word concordancing methods assigned the email wrong. This
evidence, based on an attempt to assign such a short sample, is hardly wa-
tertight. Nonetheless, taken at face value, it indicates that letter and tag
unigrams can successfully determine Yahtzee’s authorship, and the other
kinds of evidence discussed in this section cannot.

8.1.7 What, if any is the relation between ”obvious”
and ”useful” methods of author identification?”

Going by the evidence presented here, as well as the Chaski and Forsyth
reports discussed in Chapter 2, there is little if any relation. The character
unigrams, which were most effective, based on the ”ratio” measure, in assign-
ing Yahtzee’s email to the correct corpus, presented few if any patterns that
made sense to human eyes, when the significant figures were examined. In
some ways, this is probably a very desirable feature for any linguistic ”finger-
print”: a linguistic feature that is not susceptible to metalinguistic awareness
is going to be very hard to consciously change, and is more likely to remain
stable across a variety of the subject’s writing styles. The few patterns that
were susceptible to the metalinguistic analysis could be altered easily... how
hard is it to start using ellipses? Or exclamation marks!
8.2 Which of the tested methods are the most effective means of author identification?

8.2.1 The corpora

The corpora consist of the writings of three American women, in their late twenties or early thirties, all educated to post-graduate level in college and discussing similar topics. The writers are known by the codes madp, zort, and tham. There is also a fourth set of corpora: the friends corpora. This consists of writings made by many different people, most of whom will also be possessed of the same level of linguistic sophistication and be discussing similar corpora.

8.2.2 The Input

friends1 f
friends2 f
friends3 f
madp1 m
madp2 0
tham1 0
tham2 t
tham3 t
zort1 0
zort2 z
zort3 z

The names of the corpora are madp1, madp2, tham1, tham2, etc.

8.2.3 The Task

The task assigned the different methods of author identification, is to correctly assign the corpora zort1, tham1, and madp2 to their respective authors (tham, zort and madp, in case there was any doubt) The corpora to be assigned were chosen arbitrarily out of the set available for each author.
8.2.4 The Results

Keywords 1

lj.k_key1.m0.rank Right: 1.29113804359198 Wrong: 3.76579861026574
Ratio: 2.91665064704406
tham1.t tham1 fits in category t with score 0.405491114946971
tham1.f tham1 fits in category f with score 1.31751698734713
zort1.z zort1 fits in category z with score 1.54892132389235
madp2.m madp2 fits in category m with score 1.71706686123011

This is the result from using the keywords in the file key to assign the corpora. As can be seen, all the corpora were correctly assigned. However, the false assignation of tham1 to category f was preferred over two correct assignations, casting some doubt on the extent of the reliability of this method.

The words in file key are:

for
the
and
or
from
to
is
are
do
in
a
as

Keywords 2

lj.k_keywords1.m0.rank Right: 1.20821577920937 Wrong: 3.80796306806442
Ratio: 3.15172433069551
tham1.t tham1 fits in category t with score 0.187981801685173
tham1.f tham1 fits in category f with score 0.495493015957792
zort1.z zort1 fits in category z with score 0.713799161264302
madp2.m madp2 fits in category m with score 0.883697965880647
CHAPTER 8. RESULTS AND DISCUSSION

This is the result from using the keywords in the file `keywords` to assign the corpora. As can be seen, all the corpora were correctly assigned. Again, the false assignment of `tham1` to category `f` was preferred over two correct assignments, casting some doubt on the extent of the reliability of this method. However, since the Ratio measure is higher than the previous method, this method should probably be preferred over that one.

The words in file `keywords` are:

- for
- the
- and
- or
- from
- to
- is
- are
- do
- in

Letter Unigrams, Asymmetrical, No M

<table>
<thead>
<tr>
<th>Name</th>
<th>Rank Right</th>
<th>Wrong</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lj.11.m0</code></td>
<td>2.636092</td>
<td>7.82567</td>
<td>2.968662</td>
</tr>
<tr>
<td><code>zort1.z</code></td>
<td>0.544114</td>
<td>0.75764</td>
<td>0.700569</td>
</tr>
<tr>
<td><code>madp2.m</code></td>
<td>0.997240</td>
<td>1.09512</td>
<td>1.09512</td>
</tr>
<tr>
<td><code>tham1.t</code></td>
<td>0.997240</td>
<td>1.09512</td>
<td>1.09512</td>
</tr>
<tr>
<td><code>zort1.m</code></td>
<td>0.997240</td>
<td>1.09512</td>
<td>1.09512</td>
</tr>
</tbody>
</table>

All corpora were assigned to their correct authors. No false assignments were preferred over the three correct ones. This seems like a good, reliable method. This was the case with almost every letter unigram method tested, and every combination of factors that was combined with "ll". The letter unigram method seems to be a remarkably stable method of author identification.

The best methods found, as measured by the "ratio" measure, were the two following:
Letter Unigrams, Symmetrical, Count Whitespace, m=50

lj.ls1.m50.x.rank Right: 2.69749402960883 Wrong: 10.4178534901485
Ratio: 3.86204876666925
zort1.z zort1 fits in category z with score 0.0760932431153265
madp2.m madp2 fits in category m with score 0.0881704173865116
tham1.t tham1 fits in category t with score 0.161033425565944
zort1.m zort1 fits in category m with score 0.173766814118212

This method is the best out of all of those tested, by the ratio measure.

Letter Unigrams, Symmetrical, m=50

lj.l1.m50.x.rank Right: 2.66744905784323 Wrong: 10.2372457059637
Ratio: 3.83784112984749
zort1.z zort1 fits in category z with score 0.158824205301786
madp2.m madp2 fits in category m with score 0.170516432534913
tham1.t tham1 fits in category t with score 0.330198864264461
madp2.z madp2 fits in category z with score 0.342745820298499

This method is nearly as good as the one preceding it, by the ratio measure.

Letter Unigrams, Asymmetrical, m=50

lj.l1.m50.rank Right: 2.78750274619552 Wrong: 7.99699553949727 Ratio: 2.86887449722224
zort1.z zort1 fits in category z with score 0.314214427490408
madp2.m madp2 fits in category m with score 0.325084287716975
madp2.z madp2 fits in category z with score 0.636925268479305
tham1.t tham1 fits in category t with score 0.668516603820807

Surprisingly, since this method is so similar to the one preceding it, which was one of the best found, this method falls down on the stability measure, with a false assignment preferred to one of the real ones. Comparing this method to the one above is a strong signal that the symmetrical method is to be preferred to the asymmetrical one.
CHAPTER 8. RESULTS AND DISCUSSION

Letter Bigrams, m=0, Asymmetrical

lj.12.m0.rank Right: 1.31823629628382 Wrong: 2.01811660614234 Ratio: 1.53092174129291
zort1.z zort1 fits in category z with score 0.142611776078419
tham1.t tham1 fits in category t with score 0.157987042380251
madp2.m madp2 fits in category m with score 0.16411310009894
tham1.f tham1 fits in category f with score 0.17377709533825

This method assigns all the corpora correctly, and passes the initial stability criterion. However, since many other letter bigram methods assigned corpora wrongly, it is doubtful this method really is stable.

Letter Bigrams, m=50, Symmetrical

lj.12.m50.x.rank Right: 1.76538491924171 Wrong: 3.39325954422638 Ratio: 1.92210747199761
tham1.f tham1 fits in category f with score 0.0724757940025708
tham1.t tham1 fits in category t with score 0.073933343978048
tham1.m tham1 fits in category m with score 0.097173094042365
madp2.f madp2 fits in category f with score 0.100407641485275

This particular method demonstrates the flaw at the heart of the ”ratio” measure. This method scores much higher than the preceding one on the ”ratio” score, but assigns everything wrongly. A better statistical measure is certainly needed. One candidate for this method is the Wilcoxon Matched pairs calculation technique: see [17] and [20]. However, this method was not encountered in the course of research until time factors prevented any implementation of it as a way of measuring results. This method would also have provided significance scores for the author assignations; it is a definite candidate for any future work.

Letter Trigrams

These were found to be even more unreliable than letter bigrams.

Consonants Only

It was decided to make a consonants only option to see if any kind of linguistic motivation could be assigned to the letter-unigram method. However the consonants only option made little difference to the results.
CHAPTER 8. RESULTS AND DISCUSSION

Tag Unigrams

lj.t1.m0.x.rank Right: 2.60740466158853 Wrong: 7.43780137028006 Ratio: 2.85256887043712
zort1.z zort1 fits in category z with score 0.05794791982426
madp2.m madp2 fits in category m with score 0.0633458922142906
zort1.m zort1 fits in category m with score 0.0913480723627135
madp2.z madp2 fits in category z with score 0.111681484065899

It is possible that this method assigned all the authors correctly: only the topur assignations were printed and than does not feature in any of them. This method’s reliability is extremely suspect.

Tag Bigrams

lj.t2.m0.rank Right: 1.25067565599296 Wrong: 2.06221627461805 Ratio: 1.64888175822115
zort1.z zort1 fits in category z with score 0.0628791138448952
madp2.m madp2 fits in category m with score 0.0730890082681192
tham1.t tham1 fits in category t with score 0.0734176341361222
madp2.z madp2 fits in category z with score 0.0853420202690054

This method assigned all the authors correctly, and passes the intial reliability test. It is a good candidate for further investigation.

Tag Trigrams

It appears unlikely that this method will be as promising as tag bigrams.

Word Unigrams

Only the symmetrical version, with punctuation counting as a word, assigned all the corpora correctly. However, this particular method does appear quite promising.

lj.w1.m0.x.rank Right: 1.2537748677568 Wrong: 1.48301398393976 Ratio: 1.18283913809271
zort1.z zort1 fits in category z with score 0.029673197275585
tham1.t tham1 fits in category t with score 0.0301692003320216
madp2.m madp2 fits in category m with score 0.0321786896476939
tham1.f tham1 fits in category f with score 0.0323709257533332
CHAPTER 8. RESULTS AND DISCUSSION

Word Bigrams
No method tested that used word bigrams assigned all the corpora correctly.

Word Trigrams
No method tested that used word trigrams assigned all the corpora correctly.
Chapter 9

Conclusion
9.1 On the possibility of author identification

There is no question that author identification is a possibility using some of the methods outlined in this project. Several of them consistently assigned authors correctly across several corpora.

9.2 The software

A versatile set of programs has been created, that can be used both to investigate methods of author identification, and actually to identify the author of a given corpus, with a strong chance of success. The programs are modular enough that it will be easy to adapt them farther if need be.

9.3 The most effective methods

The most effective methods were the methods based on letter unigrams. The most effective method in this particular experiment is the one called by the command

\[
\text{interface } -l -n 1 -v -s -x -m 50 -f \text{it corpuslist}
\]

However, in other experiments slight variations on the same could easily prove more effective. What is hard to dispute, however, is the fact that the letter unigram method works. It even served to identify the author of a 300-word email. This backs up the conclusions of [7], strongly.

9.4 Implications

This may be a hard conclusion for many people to stomach. The letter unigram method appears to have very little linguistic/theroetical basis. Perhaps a large scale experiment to determine which letters vary most consistently between individuals, with results like those in 8.1, only many more, could shed some light on the subject. Perhaps it would be possible to compile a list of ”key letters” similar to the list of ”key words” that was used in this experiment. In the meantime, however, the motivations of this form of author identification must remain a mystery.
CHAPTER 9. CONCLUSION

9.5 Other Effective Methods

- Tag Bigrams
- Keywords
- Word Unigrams, including punctuation.
- Tag Unigrams (possibly)

All these methods, while nowhere near as effective as the letter unigrams, seem to be worthy of further investigation.

9.6 Possibilities for further work

- One possibility for further work has already been outlined in 9.4.
- Another possibility would be to take the work presented in this project, and expand it across more experiments and more corpora, to check the consistency of results.
- Multivariate analysis: take some of the measures such as those outlined and rejected by Chaski, or those input into the neural net by Merriam and Matthews, or any of the countless other potential discriminants outlined in literature, and determine which combinations of these are effective discriminants.
- This is almost an essential step if the work done in this project is to be extended: replace the “ratio” measure with a better statistical technique. As mentioned already, The Wilcoxon Matched Pairs method seems like a very good candidate.
- Possible experiment in the effects of metalinguistic awareness: invite fiction writers or linguists, people with high metalinguistic awareness, to write in the style of other authors. Compare the results to the writing both of the actual author and the imitated author.
- This is a very wide field, which I have barely scratched the surface of. The possibilities for further investigation are endless.
Bibliography


BIBLIOGRAPHY


Chapter 10

Appendix 1: Code for Concordancers

letter

#!/usr/local/bin/perl -w

use Getopt::Std;
getopts("n:o:i:vs");

$n = $opt_n;
(infile = $opt_i;
$outfile=$opt_o;
open (DATA, "$infile");
while(<DATA>){
    if ($opt_s){
s/\s+/ /g;# replace multiple letters with single letters
    }else{
s/\s//g;}
    if ($opt_v){
s/[AEIOUaeiou]/g;} #if -v
    #s/\*\*//g;
    for $letter (split//)
    {
        if ($letter eq " "){push @letters, "SPACE";}#if -s
else{
    push @letters, $letter;
}
}
}

open (OUT, "> $outfile");
#foreach $letter (@letters) {print" $letter\n";}
for $i (0 ..($#letters-$n+1)){
    $tmp=join( ' ',@letters[$i..($i+$n-1)]);
    #print $tmp;
    $hashcount{$tmp}++;
}

print OUT ($#letters-$n+1);
print OUT"\n";
foreach $ngram (sort{ $hashcount{$b} <=> $hashcount{$a} } keys %hashcount ){
    $var=$hashcount{$ngram}/($#letters+1);
    print OUT "$ngram\t$hashcount{$ngram}\t$var\n";
}
close(OUT);

concordance

#!/usr/local/bin/perl -w
#http://www.perldoc.com/perl5.6/pod/perlre.html
#http://www-2.cs.cmu.edu/cgi-bin/perl-man
use Getopt::Std;
getopts("n:o:i:");

    $n = $opt_n;
    $infile = $opt_i;
    $outfile=$opt_o;

    open (DATA, "$infile");
    open (OUT, "> $outfile");
while(<DATA>){
  #s/[^a-z|A-Z|0-9|\s]/g;
  s/\[\^\w|\s]/g;
  $=_lc($_);
  push @words,split;
}

#foreach (@words){print "$_\n";}

#words now in array
$tmp=$#words-$n+2;
print OUT "$tmp\n";
for $i (0 ..($#words-$n+1)){
  $tmp=join(’,’,@words[$i..($i+$n-1)]);
  #print "$tmp\n";
  $hashcount{$tmp}++;
  #count 'em
  push @ {$hashwordnumber{$tmp} }, $i; #make note of word number
}

# 2 hashes of ngrams

foreach $ngram ( sort { $hashcount{$b}<$hashcount{$a}|| $hashwordnumber{$a}[0]<=>$hashwordnumber{$b}[0] } keys %hashcount){
  $tmp=join(’,’,@{$hashwordnumber{$ngram}});
  print OUT "$ngram\t$hashcount{$ngram}\t$tmp\n"
}

close(OUT);

punconcord

#!/usr/local/bin/perl -w
#http://www.perldoc.com/perl5.6/pod/perlre.html
CHAPTER 10. APPENDIX 1: CODE FOR CONCORDANCERS

88

# http://www-2.cs.cmu.edu/cgi-bin/perl-man
use Getopt::Std;
getopts("n:o:i:");

$n = $opt_n;
(infile = $opt_i;
$outfile=$opt_o;

open (DATA, "$infile");
open (OUT, "> $outfile");

while(<DATA>){
$s/([^\w|\s\s])/ $1 /g;
#$s/[~a-zA-Z0-9|\s]/g;
#$s/[\~\w,.;:'"! '/(\()|\s]/g;
#$s/,/comma/g;
#$s./fullstop/g;
#$s;/semicolon/g;
#$s::colon/g;
#$s/"/doublequote/g;

$_=lc($_);
push @words,split;
}

#foreach (@words){print "$_\n";}

# words now in array
$tmp=$#words-$n+2;
print OUT "$tmp\n";
for $i (0 ..($#words-$n+1)){
.tmp=join(’,’,@words[$i..($i+$n-1)]);
# print "$tmp\n";
$hashcount{$tmp}++; # count ’em
push @ { $hashwordnumber{$tmp} }, $i;  # make note of word number
}

# 2 hashes of ngrams

foreach $ngram ( sort {$hashcount{$b}<=$hashcount{$a} || $hashwordnumber{$a}[0]<=$hashwordnumber{$b}[0]} keys %hashcount ) {
    $tmp = join(',', @{$hashwordnumber{$ngram}});
    print OUT "$ngram\t$hashcount{$ngram}\t$tmp\n"
}
# DONE!!!
close(OUT);

tagconc.pl

#!/usr/local/bin/perl -w
# http://www.perldoc.com/perl5.6/pod/perlre.html
# http://www-2.cs.cmu.edu/cgi-bin/perl-man
use Getopt::Std;
getopts("n:o:i:");

$n = $opt_n;
(infile = $opt_i;
$outfile = $opt_o;

open (DATA, "$infile");

while(<DATA>){
    @line = split;
    push @tags, $line[1];
}
#foreach (@words) {print "$_\n";}
open(OUT, "> $outfile");
# words now in array
$tmp =$#tags-$n+1;
print OUT "$tmp\n";
for $i (0 ..($#tags-$n+1)){
$tmp = join(’ ’, @tags[$i..($i+$n-1)]);
# print "$tmp\n";
$hashcount{$tmp}++; # count ’em
push @{$hashwordnumber{$tmp}}, $i; # make note of word number }

# 2 hashes of ngrams

foreach $ngram (sort {$hashcount{$b}<$hashcount{$a}||
$hashwordnumber{$a}[0]<=$hashwordnumber{$b}[0]}
keys %hashcount){
$tmp = join(‘,’, @{$hashwordnumber{$ngram}});
print OUT "$ngram\t$hashcount{$ngram}\t$tmp\n"
}
close(OUT);

keywords

#!/usr/local/bin/perl -w
# http://www.perldoc.com/perl5.6/pod/perlre.html
# http://www-2.cs.cmu.edu/cgi-bin/perl-man
use Getopt::Std;
getopts(“o:i:k:”);

$inifle = $opt_i;
$outfile=$opt_o;
$keyfile=$opt_k;

open (DATA, "$infile");
open (OUT, "> $outfile");
open (FILELIST, "$keyfile");

while (<FILELIST>){
    chomp;
    if (defined($_)){$_=lc($_); $keywords{$_}++;}
}
close (FILELIST);

while(<DATA>){
    #s/[^a-z|A-Z|0-9|\s]///g;
    s/[^\w|\s]///g;
    $_=lc($_);
    #push @words,split;
    for $word (split){
        $wordcount++;
        if (exists $keywords{$word}){
            $hashcount{$word}++;
        }
    }
}

print OUT "$wordcount\n";
#foreach (@words){print "$_\n";}

foreach $word ( sort { $hashcount{ $b } <=> $hashcount{ $a } } keys %hashcount ){

    print OUT "$word\t$hashcount{ $word }\n"
}
#DONE!!!
close(OUT);
Chapter 11

Appendix 2: Code for Other Programs

multicdf

#!/usr/bin/perl
use Getopt::Std;

#$c1= "ARGV[0]";
#$c2= "ARGV[1]";
#if ($#ARGV==2){$nARGV[2];}else{$n=0};
#print $#ARGV,$n;
goptopt("f:o:n:m:b");
#print "$opt_a heello there \n";
if (defined $opt_n) {$n=$opt_n;}else {$n=2};
if (defined $opt_m) {$m=$opt_m;}else {$m=0};
if (defined $opt_o) {$output=$opt_o;}else {$output="cbdfout"};
$tableout="$output.tables";
 infile=$opt_f;
# print "$n $m $output $opt_p $opt_m $infile\n";
#if($opt_b){$path="/tmp/";}else{$path="/"};
 if($opt_b){$path="/";}else{$path="/tmp/"};
 open (FILELIST, $infile);
$iLoop=0;
while (<FILELIST>){
    chomp;
    if (defined($_)){@line=split;push @files,$line[0];
        $iloop++;
    }
    close(FILELIST);
    $iloop=0;
for ($ii=0;$ii<=$#files;$ii++){ # print $ii,$files[$ii]," line 31 \n"
    $open=$files[$ii];
    open(DATA,"$open");
    $corpcount[$ii]=<DATA>;
    chomp($corpcount[$ii]);
    #print "$ii line 36\n"
    while(<DATA>){
        chop;
        @line=split;
        $tmp=join ' ',@line[0..($n-1)];
        #print "$ii\n"
        $corpus{$ii}{$tmp}=$line[$n];
    } #end while (<DATA>)
    close(DATA);
}#end for filelist
#all data is now read in
open(OUT,"> $output");
open(TABLE, " > $path$tableout");
print OUT "m=$m\n";
for ($ii=0;$ii<=$#files;$ii++){
    for ($jj=0;$jj<=$#files;$jj++){
        if ($ii==$jj){next};
        $chisum=0;
        # print "$files[$ii] $files[$jj] \n";
        print TABLE "$files[$ii] $files[$jj] \n";
        foreach $key (sort {$corpus{$ii}{$b} <=> $corpus{$ii}{$a} } keys %{$corpus{$ii}}){
            #sorted in descending order of frequency in first corpus
            # observed frequencies
            $ob[0]=corpus{$ii}{$key};
            if (exists(corpus{$jj}{$key})){
                $ob[1]=corpus{$jj}{$key};
            }else{
                $ob[1]=0;
            }
            #row and column totals
            $r1=$ob[1]+$ob[0];
            #column totals are corpcount values
            $NN=corpcount[$ii]+$corpcount[$jj];
            #expected frequency
            $e[0]=$r1*corpcount[$ii]/$NN;
            $e[1]=$r1*corpcount[$jj]/$NN;
            for ($j=0;$j<2;$j++){
                # print "$key $j $ob[$j] $e[$j] \n";
                $chi[$j]=(($ob[$j]-$e[$j])*(($ob[$j]-$e[$j])/$e[$j];
                #print "$key $ob[$j] $e[$j] $chi[$j] \n";
                $chisum+=($chi[$j];
            }
        printf TABLE ("%.20s o1 %.3f o2 %.3f e1 %.3f e2 %.3f 1(oe)^2/e %.3f 2(oe)^2/e %.3f sum %.3f
",$key,$ob[0],$ob[1],$e[0],$e[1],$chi[0],$chi[1],$chisum);
$iloop++; 
if ($m && $iloop>=$m){last;}

}#end of foreach loop

$cbdf=$chisum/($iloop-1); 
$iloop=0; 
$resultshash{"$files[$ii] $files[$jj]"}=$cbdf;  
# print "$files[$ii] $files[$jj] $cbdf\n"; 
# print OUT "$files[$ii] $files[$jj] $cbdf\n";
print TABLE "$files[$ii] $files[$jj] $cbdf for m=$m\n"
}}#end j, i loops

foreach $key (sort {$resultshash{$b} <=> $resultshash{$a}} keys %resultshash){
print "$key $resultshash{$key}\n";
print OUT "$key $resultshash{$key}\n";
}

multicdf.new

#!/usr/bin/perl -w
use Getopt::Std;
#rewrite concordancing programs using getopts

getopts("f:n:m:wltpcxvbks:z");

if (defined $opt_f || die "no corpus list supplied");
if (defined $opt_n) {$n=$opt_n;}
if (defined $opt_m) {$m=$opt_m;}

open (FILELIST, $infile);

while (<FILELIST>){
chomp;
if (defined($_)){
@line=split;push @files,$line[0];
$iloop++;}
close(FILELIST);
if (not $opt_c) {
    # print "woo-hoo\n";
    if ($opt_p || $opt_t) {
        $x = "t";
    } else if ($opt_l) {
        $x = "l";
        if ($opt_v) { $x = "lv"; }
        if ($opt_s) { $x = join(’,’, $x, "s"); }
    } else if ($opt_z) {
        $x = "w";
        if ($opt_v) { $x = "w!"; }
    }
    foreach $file (@files) {
        push @newfiles, "$file.t$n"
    }
    print @newfiles;
} else if ($opt_t) {
    $x = "t";
    foreach $file (@files) {
        system("/users/ug/mccombn/cmd/tree-tagger-english $file > $file.tagged.temp");
        system("perl", "/users/ug/mccombn/FYP/postprocess.pl", "-i", "$file.tagged.temp", "-o";
        push @newfiles, "$file.t$n"
    }
} elsif ($opt_p) {
    $x = "t";
    foreach $file (@files) {
        system("perl", "/users/ug/mccombn/FYP/tagconc.pl", "-n", "$n", "-i", "$file.tagged", "-o";
        push @newfiles, "$file.t$n"
    }
} elsif ($opt_l) {
    $x = "l";
    foreach $file (@files) {
        system("perl", "/users/ug/mccombn/FYP/tagconc.pl", "-n", "$n", "-i", "$file.tagged", "-o";
        push @newfiles, "$file.t$n"
    }
} elseif ($opt_z) {
    $x = "w";
    if ($opt_v) { $x = "w!"; }
    foreach $file (@files) {
        push @newfiles, "$file.t$n"
    }
    print @newfiles;
} else {
    $x = "w";
    if ($opt_v) { $x = "w!"; }
    foreach $file (@files) {
        push @newfiles, "$file.t$n"
    }
    print @newfiles;
}
@list=();
#$outfile ="$file.l$n";
if ($opt_v){$x=join('',$x,"v");push @list,"-v";  }
  if ($opt_s){$x=join('',$x,"s");push @list,"-s";  }
unshift(@list,"perl","/users/ug/mccombn/FYP/letter","-n","n","-i","file","-o","file.n");
push @newfiles,"file.n$";
  system(@list);
}
}elsif(defined($opt_k)){
  foreach $file (@files){
    $x="k_$opt_k";
    $n=1;
    system("perl","/users/ug/mccombn/FYP/keywords","-i","file","-o","file.k_$opt_k");
push @newfiles,"file.k_$opt_k";}
}
}else{
  if ($opt_z){$x="w!";$prog="/users/ug/mccombn/FYP/punconcord";}
  else{$x="w";$prog="/users/ug/mccombn/FYP/concordance";}
print @files;
  foreach $file (@files){
    $outfile="$file.$x$n";
    system("perl","$prog","-n","n","-i","file","-o","$file","-o","$outfile");
push @newfiles,"$file.$x$n";
    #print @newfiles;
}#foreach
}#else (word option)
}#end if for not c option
# print $x,$n,"\n";
open(OUT,"> $infile.$x$n.conclist");
  foreach $newfile (@newfiles){
    print OUT "$newfile\n";
    print "$newfile\n";
  }
close(OUT);
if ($opt_x){
$prog="/users/ug/mccomb/FYP/multicdf.new";
$output="$ infile.$x$n.m$m.x.rank";
} else {
$prog="/users/ug/mccomb/FYP/multicdf";
$output="$ infile.$x$n.m$m.rank";
}
@list=("perl", $prog, "-n", $n, "-m", $m, "-f", "$ infile.$x$n.conclist", "-o", $output);
if ($opt_b) {push @list, "-b"};
system(@list);

Postprocess.pl

#!/usr/bin/perl -w
use Getopt::Std;
getopts("r:f:n:");

# %corpora =
# madp1 =>1, madp2 =>1,
# tham1 =>2, tham2 =>2, tham3 =>2,
# zort1 =>3, zort2 =>3, zort3 =>3,
# friends1 =>4, friends2 =>4, friends3 =>4);
# print %corpora;
unless (defined($opt_n)) {$opt_n=4;}
open (CORPS, $opt_f) || die "no corpus list supplied";
while (<CORPS>) {
  chomp;

  if (defined($_)) { @line = split; $corpora{$_[0]} = $_[1]; }
}
close(CORPS);

open(FILELIST, $opt_r) || die "no results list supplied";
while (<FILELIST>) {
  chomp;
  if (defined($_)) { push @files, $_; }
}
close(FILELIST);
# print @files;
for ($ii=0;$ii<=$#files;$ii++){
    $open=$files[$ii];
    print "$open";
    open(DATA,"$open");
    # print $open;
    $tmp=<DATA>;
    $right=0,$wrong=0,$rightcount=0,$wrongcount=0;
    #print "line 26";
    while (<DATA>){
        chomp;
        @line=split;
        #print $line[0];
        @tmp=split (/\./,$line[0]);
        # print @tmp;
        $line[0]=$tmp[0];
        @tmp=split /\./,$line[1];
        $line[1]=$tmp[0];
        # print $line[0],$line[1];
        #print $corpora{madp1};
        if (($corpora{$line[0]}eq"0") && $corpora{$line[1]}eq"0"){
            next;
        } # is this the right thing to do?
        elsif ($corpora{$line[0]}eq"0"){
            $score{"$line[0].$corpora{$line[1]}"}+=$line[2];
            $scorecount{"$line[0].$corpora{$line[1]}"}++;
        }
        elsif ($corpora{$line[1]}eq"0"){
            $score{"$line[1].$corpora{$line[0]}"}+=$line[2];
            $scorecount{"$line[1].$corpora{$line[0]}"}++;
        }
        elsif($corpora{$line[0]}eq $corpora{$line[1]}){
            $right+==$line[2];
            break; // or continue;
$rightcount++;
} else{

$wrong+=$line[2];
$wrongcount++;
}
}
#
print $right,$wrong,$rightcount,$wrongcount,"\n";
$rightav=$right/$rightcount;
$wrongav=$wrong/$wrongcount;
$ratio=$wrongav/$rightav;

print "\tRight: $rightav\tWrong: $wrongav\tRatio: $ratio\n";
if (scalar(keys %score)){
  for $key (keys %score){
    $score{$key}=($score{$key}/$scorecount{$key});
  }
  $loop=0;
  for $key (sort {$score{$a}<=>$score{$b}} keys %score){

    print $key;
    @tmp=split (/\./,$key);

    print "\t\t$tmp[0] fits in category $tmp[1] with score $score{$key}\n";

    $loop++;
    unless ($loop < $opt_n) {last;}
  }
}
close(DATA);
}
Chapter 12

Appendix 3: Full results of LJ experiment

lj.k_key1.m0.rank Right: 1.29113804359198 Wrong: 3.76579861026574
Ratio: 2.91665064704406
tham1.t tham1 fits in category t with score 0.405491114946971
tham1.f tham1 fits in category f with score 1.31751698734713
zort1.z zort1 fits in category z with score 1.54892132389235
madp2.m madp2 fits in category m with score 1.71706686123011
lj.k_key1.m0.x.rank Right: 1.29113804359198 Wrong: 3.76579861026574
Ratio: 2.91665064704406
tham1.t tham1 fits in category t with score 0.202745557473485
tham1.f tham1 fits in category f with score 0.585563105487612
zort1.z zort1 fits in category z with score 0.774460661946175
zort1.f zort1 fits in category f with score 0.981666505813225
lj.k_keywords1.m0.rank Right: 1.20821577920937 Wrong: 3.80796306806442
Ratio: 3.15172433069551
tham1.t tham1 fits in category t with score 0.187981801685173
tham1.f tham1 fits in category f with score 0.495493015957792
zort1.z zort1 fits in category z with score 0.713799161264302
madp2.m madp2 fits in category m with score 0.883697965880647
lj.l1.m0.rank Right: 2.63609217366066 Wrong: 7.82566724161618 Ratio: 2.96866221895
zort1.z zort1 fits in category z with score 0.544114746663495
madp2.m madp2 fits in category m with score 0.757643771786727
tham1.t tham1 fits in category t with score 0.997240509733984
zort1.m zort1 fits in category m with score 1.09512491150071
lj.11.m0.x.rank Right: 2.70951996441649 Wrong: 8.1393272941999 Ratio: 3.00397391460145
zort1.z zort1 fits in category z with score 0.254449499683505
madp2.m madp2 fits in category m with score 0.378179097127396
tham1.t tham1 fits in category t with score 0.505373208420257
madp2.z madp2 fits in category z with score 0.528655036628638

lj.11.m100.x.rank Right: 2.70951996441649 Wrong: 8.1393272941999 Ratio: 3.00397391460145
zort1.z zort1 fits in category z with score 0.210084819330894
madp2.m madp2 fits in category m with score 0.29399645581776
tham1.t tham1 fits in category t with score 0.421894668522799
zort1.m zort1 fits in category m with score 0.43345028540878

lj.11.m40.x.rank Right: 2.84468827792103 Wrong: 9.24492769666255 Ratio: 3.24989130387917
madp2.m madp2 fits in category m with score 0.195408760932205
zort1.z zort1 fits in category z with score 0.196353989292506
madp2.z madp2 fits in category z with score 0.381334989292506
tham1.t tham1 fits in category t with score 0.43740364126286

lj.11.m50.rank Right: 2.78750274619552 Wrong: 7.99699553949727 Ratio: 2.86887449722224
zort1.z zort1 fits in category z with score 0.314214427490408
madp2.m madp2 fits in category m with score 0.325084287716975
madp2.z madp2 fits in category z with score 0.636925268479305
tham1.t tham1 fits in category t with score 0.668516603820807

lj.11.m50.x.rank Right: 2.66744905784323 Wrong: 10.2372457059637 Ratio: 3.73750023350143
madp2.m madp2 fits in category m with score 0.11997866669756
madp2.z madp2 fits in category z with score 0.129209433794379
tham1.t tham1 fits in category t with score 0.25676944593996

lj.11.m60.rank Right: 2.7744992879052 Wrong: 8.25910463333327 Ratio: 2.97679104454521
zort1.z zort1 fits in category z with score 0.236102608269641
madp2.m madp2 fits in category m with score 0.254617352902077
madp2.z madp2 fits in category z with score 0.492472611364675
tham1.t tham1 fits in category t with score 0.498236291542248

lj.11.m60.x.rank Right: 2.59661776124111 Wrong: 9.70485948895261 Ratio: 3.73750023350143
zort1.z zort1 fits in category z with score 0.11997866669756
madp2.m madp2 fits in category m with score 0.129209433794379
madp2.z madp2 fits in category z with score 0.25676944593996
tham1.t tham1 fits in category t with score 0.262914232083859
CHAPTER 12. APPENDIX 3: FULL RESULTS OF LJ EXPERIMENT

lj.12.m0.rank Right: 1.31823629628382 Wrong: 2.0181660614234 Ratio: 1.53092174129291
zort1.z zort1 fits in category z with score 0.142611776078419
than1.t than1 fits in category t with score 0.157987042380251
madp2.m madp2 fits in category m with score 0.16411310009894
than1.f than1 fits in category f with score 0.17377709533825
lj.12.m0.x.rank Right: 1.35817286438622 Wrong: 1.95815977937955 Ratio: 1.44176034636392
zort1.z zort1 fits in category z with score 0.0724576156791666
than1.t than1 fits in category t with score 0.0776672707287184
than1.f than1 fits in category f with score 0.0822989714289742
madp2.m madp2 fits in category m with score 0.0841328925909226
lj.12.m50.rank Right: 1.92705871791183 Wrong: 3.54544380646507 Ratio: 1.83982136792746
than1.t than1 fits in category t with score 0.155782754390983
than1.f than1 fits in category f with score 0.16228828481657
than1.m than1 fits in category m with score 0.181902152833811
madp2.m madp2 fits in category m with score 0.203104726465854
lj.12.m50.x.rank Right: 1.76538491924171 Wrong: 3.39325954422638 Ratio: 1.92210747199761
than1.f than1 fits in category f with score 0.0724757940025708
than1.t than1 fits in category t with score 0.0739333343978048
than1.m than1 fits in category m with score 0.09713094042365
madp2.f madp2 fits in category f with score 0.100407641485275
lj.13.m50.rank Right: 2.93068661929069 Wrong: 6.26249011880701 Ratio: 2.13686788535675
madp2.m madp2 fits in category m with score 0.15623805743085
than1.t than1 fits in category t with score 0.17420785940369
than1.f than1 fits in category f with score 0.201808164937529
madp2.f madp2 fits in category f with score 0.224818355262308
lj.ls1.m50.x.rank Right: 2.69749402960883 Wrong: 10.497534901485 Ratio: 3.8620487666925
zort1.z zort1 fits in category z with score 0.0760932431153265
madp2.m madp2 fits in category m with score 0.0881704173865116
than1.t than1 fits in category t with score 0.16103425565944
zort1.m zort1 fits in category m with score 0.1737668141212
lj.ls2.m50.rank Right: 2.17429488574741 Wrong: 4.97759694374542 Ratio: 2.28929248574964
zort1.z zort1 fits in category z with score 0.143981797890372
madp2.m madp2 fits in category m with score 0.181146932419108
than1.t than1 fits in category t with score 0.20135313397791
than1.m than1 fits in category m with score 0.205989805937941
lj.lv1.m50.x.rank Right: 2.77372537412623 Wrong: 10.4985284622199
CHAPTER 12. APPENDIX 3: FULL RESULTS OF LJ EXPERIMENT

Ratio: 3.7849204000219
zort1.z zort1 fits in category z with score 0.0733672757091473
madp2.m madp2 fits in category m with score 0.0820780057312249
tham1.t tham1 fits in category t with score 0.151330051246721
zort1.m zort1 fits in category m with score 0.153171620722818
lj.lvs1.m50.x.rank Right: 2.69792572541669 Wrong: 10.4780052822846
Ratio: 3.88372636932631
zort1.z zort1 fits in category z with score 0.0681867835883596
madp2.m madp2 fits in category m with score 0.0747057968703857
tham1.t tham1 fits in category t with score 0.135858285903709
zort1.m zort1 fits in category m with score 0.145834922748533
lj.lvs1.m50.x.rank Right: 2.63654965481279 Wrong: 7.495498017775012 Ratio: 2.84291934501129
zort1.z zort1 fits in category z with score 0.117040460618389
madp2.m madp2 fits in category m with score 0.1252969259175
zort1.m zort1 fits in category m with score 0.18361054900923
madp2.z madp2 fits in category z with score 0.22185593617139
lj.t1.m0.rank Right: 2.60740466158853 Wrong: 7.43780137028006 Ratio: 2.85256887043712
zort1.z zort1 fits in category z with score 0.0579847911982426
madp2.m madp2 fits in category m with score 0.0633458922142906
zort1.m zort1 fits in category m with score 0.0913480723627135
madp2.z madp2 fits in category z with score 0.11681484065899
lj.t1.m0.x.rank Right: 2.88058563818633 Wrong: 8.13158084564624 Ratio: 2.82289154602813
zort1.z zort1 fits in category z with score 0.112978476340249
madp2.m madp2 fits in category m with score 0.122240662369233
zort1.m zort1 fits in category m with score 0.167958094358338
tham1.t tham1 fits in category t with score 0.21613733984364
madp2.z madp2 fits in category z with score 0.31163289801379
lj.t1.m30.rank Right: 2.91798005298979 Wrong: 9.663278739265 Ratio: 3.31163289801379
madp2.m madp2 fits in category m with score 0.0571982750862094
zort1.z zort1 fits in category z with score 0.0616234725043579
zort1.m zort1 fits in category m with score 0.0856593673750811
tham1.t tham1 fits in category t with score 0.117179608130316
lj.t1.m20.rank Right: 1.25067565599296 Wrong: 2.06221627461805 Ratio: 1.64888175822813
zort1.z zort1 fits in category z with score 0.0628791138448952
madp2.m madp2 fits in category m with score 0.0730890088581192
tham1.t tham1 fits in category t with score 0.0734176341361222
madp2.z madp2 fits in category z with score 0.085432020690054
lj.t3.m0.rank Right: 1.02649234009204 Wrong: 1.421102234000051 Ratio: 1.38442556120000
zort1.z zort1 fits in category z with score 0.0528811936336756
CHAPTER 12. APPENDIX 3: FULL RESULTS OF LJ EXPERIMENT

tham1.f tham1 fits in category f with score 0.055847855610866
tham1.t tham1 fits in category t with score 0.0559375243793177
madp2.f madp2 fits in category f with score 0.058194220451121
lj.w!1.m0.rank Right: 1.23503851258485 Wrong: 1.86851736065776 Ratio: 1.512922343407
zort1.z zort1 fits in category z with score 0.0569399325970219
tham1.t tham1 fits in category t with score 0.0632854968784152
tham1.f tham1 fits in category f with score 0.0663582971946522
madp2.m madp2 fits in category m with score 0.0697957494962735
lj.w!1.m0.x.rank Right: 1.2537748677568 Wrong: 1.48301398393976 Ratio: 1.18283913809271
zort1.z zort1 fits in category z with score 0.029673197275585
tham1.t tham1 fits in category t with score 0.0301692003320216
madp2.m madp2 fits in category m with score 0.032178689647939
tham1.f tham1 fits in category f with score 0.032370925753332
lj.w!2.m40.rank Right: 3.29410120274486 Wrong: 7.6410461351496 Ratio: 2.31961487060039
zort1.z zort1 fits in category z with score 0.120299108299033
tham1.t tham1 fits in category t with score 0.13270927575585
tham1.f tham1 fits in category f with score 0.1750679307928
zort1.f zort1 fits in category f with score 0.1750679307928
lj.w!2.m40.x.rank Right: 3.7771387274923 Wrong: 9.19086478019189
Ratio: 2.43330359912056
tham1.t tham1 fits in category t with score 0.0625599167866397
zort1.z zort1 fits in category z with score 0.0627484811933847
zort1.f zort1 fits in category f with score 0.0938701660822891
tham1.f tham1 fits in category f with score 0.100807630497865
lj.w!1.m0.rank Right: 1.1909234866834 Wrong: 1.72638297273583 Ratio: 1.44961671518677
zort1.z zort1 fits in category z with score 0.0489379171794033
tham1.t tham1 fits in category t with score 0.0533154254617339
tham1.f tham1 fits in category f with score 0.0535001844151223
madp2.f madp2 fits in category f with score 0.0564897566799877
lj.w!1.m0.x.rank Right: 1.2168030700102 Wrong: 1.37867236489823 Ratio: 1.13328963331618
zort1.z zort1 fits in category z with score 0.025367099864554
tham1.t tham1 fits in category t with score 0.0254782934331659
tham1.f tham1 fits in category f with score 0.026537010953652
zort1.f zort1 fits in category f with score 0.026808691474497
lj.w!1.m0.x.rank Right: 1.3374491343721 Wrong: 6.00024861083306
Ratio: 4.48635196152681
tham1.f tham1 fits in category f with score 0.0318725496936029
madp2.m madp2 fits in category m with score 0.0339548915524978
tham1.t tham1 fits in category t with score 0.0468033502957417
zort1.z zort1 fits in category z with score 0.0565029003283507
lj.w1.m50.rank Right: 2.34818911539451 Wrong: 5.93835381336128 Ratio: 2.52890781855259
zort1.z zort1 fits in category z with score 0.0794120333494869
madp2.m madp2 fits in category m with score 0.100376357561918
tham1.f tham1 fits in category f with score 0.109935109728376
tham1.t tham1 fits in category t with score 0.111986409024759
lj.w2.m0.rank Right: 1.06582560992421 Wrong: 1.48785763418332 Ratio: 1.39596723922699
zort1.z zort1 fits in category z with score 0.0396894779352463
tham1.f tham1 fits in category f with score 0.0403105449915565
madp2.f madp2 fits in category f with score 0.0411607219822384
tham1.t tham1 fits in category t with score 0.042378251956843
lj.w2.m0.x.rank Right: 1.0454871710589 Wrong: 1.07124485142177 Ratio: 1.02463701236696
zort1.z zort1 fits in category z with score 0.0192532791608257
tham1.t tham1 fits in category t with score 0.0194690692721394
tham1.f tham1 fits in category f with score 0.0196056386537092
zort1.z zort1 fits in category z with score 0.019937896460414
lj.w2.m100.x.rank Right: 1.70884637451898 Wrong: 4.24936517108862 Ratio: 2.48668677176611
tham1.t tham1 fits in category t with score 0.0277495968078814
zort1.z zort1 fits in category z with score 0.0327574367504919
tham1.f tham1 fits in category f with score 0.0331510212668301
zort1.f zort1 fits in category f with score 0.0412781222366785
lj.w2.m50.rank Right: 2.22773064406824 Wrong: 4.67174858622137 Ratio: 2.0970886218729
tham1.t tham1 fits in category t with score 0.06628356565882973
tham1.f tham1 fits in category f with score 0.0780331815495409
zort1.z zort1 fits in category z with score 0.082946334713736
zort1.f zort1 fits in category f with score 0.0937840898388995
lj.w3.m50.rank Right: 2.18092168275546 Wrong: 3.84823857527943 Ratio: 1.7645216725825
tham1.f tham1 fits in category f with score 0.0628241694958655
zort1.z zort1 fits in category z with score 0.0662208096499854
tham1.t tham1 fits in category t with score 0.0681310379487342
zort1.f zort1 fits in category f with score 0.0756615264489679