Automatic Detection of Authorship Changes within Single Documents

by

Neil Graham

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Automatic Detection of Authorship Changes within Single Documents

Neil Graham

Master of Science

Graduate Department of Computer Science

University of Toronto

2000

One of the most difficult tasks facing anyone who must compile or maintain any large, collaboratively-written document is to foster a consistent style throughout. In this thesis, we explore whether it is possible to identify stylistic inconsistencies within documents even in principle, given our understanding of how style can be captured statistically.

We carry out this investigation by computing stylistic statistics on very small samples of text comprising a set of synthetic collaboratively-written documents, and using these statistics to train and test a series of neural networks. We are able to show that this method does allow us to recover the boundaries of authors' contributions. We find that time-delay neural networks, hitherto ignored in this field, are especially effective in this regard. Along the way, we observe that statistics characterizing the syntactic style of a passage appear to hold much more information for small text samples than those concerned with lexical choice or complexity.

Acknowledgements

Although it almost seems a cliche, it is unquestionably true that this document would not have been possible without the help of numerous people. Since I came to this field from the rather distant realm of mathematics, my task of understanding the background material was not inconsiderable. The fact that blindness makes independent review of printed technical material quite difficult would have caused this task to be even more arduous had it not been for the efforts of so many researchers in this field who generously provided me with the electronic source for their papers. While an itemization is almost certain to neglect someone, I thank David Holmes, Sameer Singh, Fiona Tweedie, Patrick Juola, Robert Matthews, David Mealand, Graeme Hirst, Geoffrey Hinton, Ian Lancashire, Judith Klavans, Dan Jurafsky and Bill Teahan for providing this invaluable service. Thanks also go to those authors who availed themselves of the CMPLG archive—one of a number of like projects that garner far too little support. Thanks are also due to Julia Boma-Fischer for providing a superlative reading of a critical paper and for general assistance with linguistic nomenclature.

Particularly in the last stages of the thesis, my extreme weakness in statistics showed itself starkly. The assistance of Melanie Baljko, Alex Budanitsky and Bhaskara Marthi in helping to fill this dearth of knowledge is greatly appreciated. Eric Joanis is also to be remembered for pointing out my grave omissions when applying Brill's part of speech tagger.

The computational demands of so many parts of this experiment meant that it would not have proceeded had excellent computer equipment not been available. Funding from the Natural Sciences and Engineering Research Council (NSERC) allowed for the purchase of sufficiently powerful hardware. Also to be highlighted is NSERC's patience—a quality all too rare, not just among funders but generally—in funding me directly for two years and indirectly through Graeme Hirst's research grant for an additional four months.

Finally and principally, I thank Graeme Hirst for ably and efficiently supervising this thesis. From setting the broad research objectives and suggesting an appropriate corpus to helping to get papers in usable formats, from setting up contacts with other researchers to providing high- and low-level suggestions for the betterment of the writing, Graeme's experience and above all patience has showed throughout. Most remarkably, despite this close involvement, Graeme has allowed me the widest possible freedom in conducting the course of the experiments.

Dedication

To my parents

Contents

List of Tables

Chapter 1

Purpose and Background

1.1 Purpose

Anyone who has spent much time reading manuals, reports, textbooks or other collaboratively-written documents can attest to the fact that such documents are often marked by a lack of stylistic cohesiveness that makes them not only less enjoyable to read but palpably more difficult to comprehend. Anyone whose task has been to put together such a document will undoubtedly confirm that attempting to impose a consistent style throughout is one of the hardest aspects of the job.

The goal of this thesis is to take a small step towards constructing software to assist editors in their efforts to create stylistically uniform documents. This will be done by investigating techniques for determining the boundaries between various authors' contributions in collaboratively-written documents. Such work, it is hoped, could act as the basis for software which would not only spot stylistic discrepancies but describe those discrepancies and suggest ways of remedying them. This investigation will also explore several interesting theoretical questions in statistical stylometry and in particular the application of neural networks to this field.

1.2 Terminology

1.2.1 Our "Style"

Before we can discuss our contentions more formally, we must develop some terminology. One of the most common content words in our study being "style", we first discuss in detail what we mean—and do not mean—by this highly polysemous term.

Even when confined to a strictly literary domain, style is not a simple concept to get hold of. Those involved with modern text mark-up languages, particularly those in the XML (Extensible Mark-up Language) family, use the term to describe aspects of a document's presentation that are customizable by either writer or reader. A document should come with, or refer to, one or more "style sheets", which describe how the contents are to be displayed—and, at least in theory, interpreted—on some given platform, and may be supplied by an author, the user, or even some third party. This meaning has no relevance at all for this study; should we need to refer to the presentational aspects of documents, we will use the less-fashionable term "format".

"Functional style", a term synonymous with genre, is used by some researchers in the field of genre detection; for this usage and a general proof of the difficulty experienced even by researchers in consistently using "style", see Karlgren [12]. Since we may assume that any collaboratively-written document will not contain subdocuments that could be placed in significantly different literary genres, this use of "style" is quite out of the scope of our work. Another use of "style" which is closer to our meaning is in the context of categorization: researchers often talk of "formal" versus "informal" style, or "suasive" versus "non-suasive" style. We will follow the convention of referring to this facet of documents as their "register". These classificatory schemes are germane to our purposes; a document written in part formally and in part colloquially must be regarded as highly stylistically incoherent. But, this problem is of interest to many researchers; in information retrieval, for instance, it would be highly desirable were a user able to specify the register of documents of interest. There seeming to be no reason why such techniques, when developed, should not be directly applicable to solving this aspect of our problem, we have chosen to concentrate on another, broader, meaning of style.

For us, an author's style is the product of all those elements of his/her idiolect—"personal language behaviour" [16]—that can be perceived in his/her writing. It is that elusive property of text which, in addition to (and doubtless in conjunction with) semantic cues, enables people to decide whether two passages share a common author. Style can be manifested at the level of the logical and syntactic structures prevalent in a text, or by the choice of words or extent of the diction employed. We make no claim that techniques do—or even can—exist to completely characterize a text in these deep and subjective terms. Yet, we do maintain that such a characterization can be approximated by statistical means, even for very short texts, and further that such approximated characterizations can be used to infer whether two given texts are from a document produced by a single author.

1.2.2 Specialized Terminology

We use this subsection to introduce some terminology in hopes of facilitating our descriptions of our hypotheses and our attempt to validate them. For the purposes of this thesis, a document will be viewed as consisting of a sequence of "parts". Parts correspond to the largest textual units of any given document which are known to have some unique author. The critical nature of the definition of part cannot be overestimated, since the stylometric statistics that our procedure relies upon will be computed over parts. Since paragraphs are the largest identifiable units of text in our corpus that we can be sure have unique authors, we assume here that a part will simply correspond to a paragraph. In other contexts, such as textbooks, where chapters are often written by single authors, a larger unit than a paragraph should be selected to correspond to a part; this follows from the fact that the more text contained in parts the more accurate the statistics computed from those parts. We prefer to use more general language in describing our hypotheses than simply to use paragraphs as the canonical unit of text because the type of textual unit to be considered as indivisible will depend on the corpus.

Particularly in chapter 2, we will have occasion to discuss methods employed to indicate that some particular portion of a given text has been taken from a previously-written text or has its origins with authors other than those primarily responsible for the text as a whole. We will term such subtexts as “quotations”. We will refer to punctuation marks traditionally used to demarcate quotations—such as double quotes, single quotes, etc.—as “quotation marks”. We will refer to less traditional ways of identifying quotations, such as greater-than signs prepended to each quoted line, as “quote marks”; such are commonly to be found in e-mail messages or USENET postings. We will also use “quote mark” to subsume the entire class of methods and symbols used to identify quotations.

1.3 Hypothesis and Outline

The hypothesis of this thesis is that stylometric statistics easily derivable from parts of documents can be used to determine boundaries between the contributions of different—unknown—authors if the contributions have not been edited with a view towards homogenizing their style, even if they are very small and topically similar. To state the proposition more specifically: in this experiment we attempt to show that it is possible to develop a neural network which takes as input a sequence of sets of stylometric statistics, each set computed from a paragraph of some document, and outputs a sequence of predictions regarding the location of authorship boundaries. If we are able to exhibit such a network, and show that its performance is significantly better than that of a baseline, we will have validated our hypothesis.

This experiment will involve several phases. First, we must select an appropriate corpus. Then, it will be necessary to divide each document in our corpus into parts, recording where the authorship contribution boundaries are located and ensuring that no obvious structural or domain-specific features exist which could bias the performance of our networks. We will compute stylometric statistics on each of the parts and prepare those statistics to be used to train and test various neural networks. If the results from the neural nets are not clear, it may prove desirable to develop a baseline statistical test for evaluating how well the neural net approach works.

1.4 Background

1.4.1 Stylometric Statistics and Style

There is considerable evidence in the literature to suggest that our hypothesis might prove to be valid, and before we proceed to describe our experiment, we will review some of the relevant studies. Work in "authorship attribution"—the field dedicated to developing reliable techniques for resolving questions of disputed authorship—is particularly relevant. This relevance stems from the fact that, to ascribe a disputed work to one of a set of putative authors, one must have some method for telling apart works known to have been produced by each author. Since an author's idiolect is reflected in his/her style, style can be expected to be the most reliable indicator upon which to base such judgements. Therefore, much work in this area has focused on quantifying authors' styles; most of our techniques originate from this endeavour.

Although Tweedie et al [33] inform us that Augustus de Morgan postulated as early as 1851 that differences in the lengths of words employed by authors might be used to settle questions of authorship, it was not until the 1880's that his idea was tested. Probably the best-known work in authorship attribution was carried out by Mosteller and Wallace [23], who worked on the Federalist Papers. Principally written by James Madison and Alexander Hamilton, the authorship of the vast majority of these essays, originally written to convince New Yorkers to ratify the U.S. Constitution, is uncontested.

However, on the night before his death, Hamilton claimed that twelve of the papers then believed to have been written by Madison were actually his own work. All of the essays vary between 900 and 3500 words, according to Tweedie et al [33], and thus each one represents a reasonably large sample of writing. Since all are of the same literary genre and concern the same topic, and since there are many undisputed examples of each potential author included in the set, they provide ideal conditions for testing statistical techniques of authorship attribution against standard literary methods. The results obtained by Mosteller and Wallace, and subsequently by many other researchers such as Holmes et al [8], generally concur with those arrived at through commonly-accepted literary methods, and thus lend strong support to the idea that stylometric statistics do in fact measure some real aspect of an author's writing.

The Federalist Papers are far from being the only instance in which disputed authorship has been studied using statistical methods. In his excellent paper [7], David Holmes mentions several examples. Chief among these have been attempts to determine whether certain disputed plays were produced by Shakespeare, Marlowe or Fletcher. The Bible has also attracted much interest in this field, since authorship of many portions is unclear. Other workers, such as Lancashire [16], combine stylometric statistics and traditional literary techniques while studying the works of a single author in detail.

We have noted above that the concept of literary genre, while irrelevant for our present purposes, is not distant from our notion of style. Several recent papers have examined genre from an information retrieval perspective, taking the view that it would be highly desirable were users able to make genre-specific queries, particularly on such a heterogeneous database as the Internet. Both Karlgren and Cutting [13] and Kessler et al [14] have successfully used stylometric statistics to detect meaningful genres in the Kučera and Francis Corpus (otherwise known as the Brown Corpus) [15], a set of text samples by many authors, each of approximately two thousand words, containing slightly more than one million words in total. Although these results might be slightly weakened by the fact that the samples comprising the Kučera and Francis Corpus were selected precisely to exemplify a very wide range of genres, they demonstrate clearly that stylometric statistics can be used to discriminate genres. Both groups of researchers go so far as to equate stylometric statistics with style, implicitly referring to their work as an attempt to use stylistic cues to detect genre. Using a quite different set of well-accepted stylometric statistics from either Karlgren and Cutting or Kessler et al, Mealand [21] shows a high degree of clustering among generically-similar sections of the gospel of Mark. While the problem of discriminating between genres is not one we address, these results are clear evidence that stylometric statistics are useful not only for discriminating among some small set of authors, but also for capturing deep facets of text.

All this research stands in support of the idea that stylometric statistics are, in some measure at least, capable of capturing some aspect of style. But, while they all rely on large samples of text, it is small samples that attract our interest in this study. In the next subsection we describe research using smaller sample sizes.

1.4.2 Work on Small Corpora

Like ourselves, other researchers have wondered whether such good results as are noted in the preceding subsection would be obtainable from smaller samples. One of the most interesting studies along this line, performed by Glover and Hirst [5], had subjects write summaries of two halves of a TV program, creating a set of artificial collaboratively-written documents by randomly matching the beginning and ending summaries of different authors. This study showed that stylometric statistics could flag collaboratively-written and non-collaboratively-written documents with reasonable probability, despite the fact that the texts involved were almost universally less than five hundred words in length.

Recent work by Patrick Juola [11] demonstrates a novel technique that appears to hold particular promise. With this technique, described in detail in section 3.2 below, Juola uses samples as small as five hundred characters to correctly classify all the disputed Federalist Papers.

Stamatatos et al [27] describe a complex but fascinating method for ascribing the authorship of small samples of modern Greek text. In this method, particular sentence- and chunk-boundary detectors are combined with a particular multi-pass parser [28] and are used to generate a rich set of statistics on the syntax of a given document; for example, the proportion of words not parsed after each pass of the parser is computed. Armed with these statistics computed on a small sample of newspaper articles written by a known set of authors, principal component analysis is used to group articles by author. While far from perfect, the results thus obtained are very encouraging in that for most authors, the majority of their articles that were used in the sample are present in a single cluster. Though no controls for genre were applied, and the focus of this work is more classically authorship-attributive than ours, it is nonetheless extremely encouraging.

1.4.3 Stylometric Statistics and Neural Networks

Most of these studies have used fairly conventional statistical techniques; however, the use of neural nets in conjunction with stylometric statistics appears to be becoming increasingly popular. Tweedie et al [32] present an extensive review of the uses of neural nets in stylometry. Their own work [33] in this connection involves the Federalist Papers. By training a single network with a small hidden layer on a subset of the function words originally used in the study by Mosteller and Wallace, Tweedie et al not only are able to correctly classify the disputed papers, but make interesting conjectures regarding three papers thought to have been jointly authored by both Hamilton and Madison.

The pioneering work in the application of neural nets to stylometry, according to Tweedie et al [32], was undertaken by Matthews and Merriam in papers such as [20]. In this paper, a very small set of function word frequencies is used as input to a multilayer perceptron (a neural net having a hidden layer) to examine four plays that have been attributed both to Shakespeare and John Fletcher. A large corpus of undisputed plays by both authors exists to train this network, and the results of the study on sections of the disputed works prove to be highly interesting, correlating reasonably well with accepted scholarship.

Neural nets have also found recent popularity in the genre detection community. Building on the results published by Karlgren and Cutting, Kessler et al [14] have used neural nets to attempt to place samples from the Brown corpus into genres. Unlike Karlgren and Cutting, Kessler et al have preferred to define their own genres, and have implemented a much more well-motivated approach to the problem of defining precisely what a genre is. Thus, the improvement in results that they observe comes as little surprise. They also compare both a linear perceptron and a multilayer perceptron approach with the output produced by logistic regression analysis—a technique considerably more powerful than the simple discriminant analysis performed by Karlgren and Cutting. Both neural nets perform better in most cases than logistic regression analysis.

So clearly our hypothesis rests on a substantial body of work, not only with respect to the usefulness of neural nets in the field, but to the notion that, even for small samples of writing, aspects of auctorial style can in fact be captured statistically. It is equally clear that we propose to break new ground in a number of directions. First, we are training our neural nets not on large corpora produced by a small number of known authors as was done by Matthews and Merriam or Tweedie et al, but on a large corpus written by a large number of unknown authors. Second, we propose to compute stylometric statistics on a large number of very small samples of work. This is important since most stylometric statistics are thought to be unreliable on small samples; [7]. Third, our neural net architectures will of necessity have to be different than those used by previous workers in this area, since we require our nets to tell us whether two samples are different, rather than to fit a single sample into one of a set of classes. Finally, we propose to use some stylometric statistics—such as distribution of punctuation—that have been widely ignored in the literature because researchers have felt they are prone to error [7]; we discuss this point further in section 3.6. All this shows that while our basic tenets are well-founded, there are many ways in which this research could prove ground-breaking.

Chapter 2

Selection and Initial Processing of the Corpus

2.1 Selecting the Corpus

For the purposes of this experiment, we need a corpus of documents satisfying several criteria. First, the corpus must be large—containing at least many hundreds of examples of small samples of text produced by many different authors. While one of the advantages of neural networks is their resistance to overfitting to small amounts of data, it is certainly true that the more data that are available the better the trained networks will be. Second, since some of our stylometric statistics will be computed from the output of part of speech taggers, which perform poorly on poorly-written text (i.e., text with many typing or grammatical errors), we must insist that our corpus contain writing of at least reasonable quality. Although our techniques are directed towards characterizing the style of authors, we must concede that we have little evidence to support the contention that these techniques are entirely independent of topic. That is, a priori we cannot state that our techniques are no less likely to successfully distinguish writings on closely related topics than pieces discussing widely divergent topics. Therefore, to test our technique thoroughly we would like our corpus to be such that authors are writing about similar issues from a similar perspective. Should we be successful, it would then be reasonable to assume that our results would generalize to other domains.

Since it satisfies most of these requirements, we have chosen to use the Risks corpus[1] as our source of documents. This corpus represents over six megabytes of unprocessed text—very nearly one million words—and is mostly composed of small chunks. Informal analysis indicates that several dozen authors at the very least have made contributions to this corpus. Since this is a moderated forum, only posts with reasonably good writing quality are included. We have some measure of topic control because of the fact that most postings tend to relate to issues of privacy, security, or software quality.

2.2 Initial Processing: Overview

One of the drawbacks of using a newsgroup archive such as the Risks corpus is that considerable effort needs to be expended to reduce the archive to a standard form from which meaningful stylometric statistics can be computed. Hence, the first step in the experiment was to write programs to perform this reduction. We discuss this effort in detail in the remainder of this chapter.

Since the purpose of this experiment is to explore whether sets of stylometric statistics can be used to determine the boundaries of authors' contributions, any structural characteristics of the corpus that correlate with those boundaries and any features that, in some sense, taint the style of a particular contribution must be recognized and if possible eliminated. In this corpus, article headers, authors' "signatures", and text quoted within articles all fall into these categories. Given that in this experiment parts will correspond to paragraphs, to allow stylometric statistics for parts to be easily computed it is

[1] *Forum on Risks to the Public in Computers and Related Systems (*`comp.risks`*), ACM Committee on Computers and Public Policy, Peter G. Neumann, moderator. We use every digest of this newsgroup from Apr. 5, 1996 through Apr. 1, 1999.*

also imperative that paragraphs be formatted consistently throughout the corpus. As we perform these modifications, we must maintain a record of exactly where the boundaries between authors' contributions occur so that we have a basis for training and testing the neural nets that we will describe in chapter 5 below. In the next few sections, we provide a rationale and detailed descriptions of the steps taken to achieve each of these goals, and give as complete a picture as possible of the successes and limitations of the actions we have taken.

Before continuing with these descriptions, it will be helpful to define precisely several terms and to describe the architecture common to all the programs used to carry out the procedure. Our corpus is divided into over two hundred files; virtually every file conforms to a very precise format. Each file begins with a header, which we will refer to as the "file header", which contains such introductory information as the issue number, a table of contents of the articles contained in the file, and often the file's transmission record (most of the files having been taken from a private e-mail archive, and so having passed from the list over the Internet to the archive.) The file header is followed by one or more "articles"—or messages composed entirely by one author, possibly with annotations (from the group moderator, for example) and embedded quotations by other authors. Each article has its own "article header", described in the next section. The rest of the article is composed of the "article body". With the exception of one file, every file ends with a characteristic "file trailer", that has the form of an article but contains exactly the same text, describing the `comp.risks` newsgroup/archive/digest, in every file. We define a "contribution" as all the text within a single article body that was written by a single author. Fortunately, collaborative writing appears to be highly unusual in this corpus. Hence, the assumption that there is an onto mapping between the set of contributions and some set of authors is not unrealistic. It is of course quite possible for a single article body to contain multiple contributions—through the author of the article having quoted other authors, for example. The last element of an article may be a “signature”—text giving information about the author, such as his/her name, affiliation, e-mail address, location, etc. We discuss signatures, and our interest in them, extensively below in section 2.8.

All the programs—or more accurately scripts—used in this phase were written in Perl. This language was chosen principally for its excellent pattern-matching and string-handling facilities, as well as the ease with which one can use it to construct “filters”. By a filter, we mean a program that accepts a file’s contents as its standard input, performs changes to the contents, and outputs the altered text on its standard output. With the exception of the index-building script described in section 2.9 below, all the scripts used in this phase were filters.

The approach taken in these scripts is to read the input line by line, process it in some way—possibly using temporary arrays—then store it in an “output array”. The contents of this output array are then sent to standard output after all available input has been read. While this approach requires each processed file to reside entirely in memory at some point, and also requires at least twice the number of memory copies that would have been needed had the processed output simply been sent to standard output as soon as processing was complete, the current approach permits output of the filters to be piped one to another; simply sending output to standard output before completing reading on standard input either results in processes blocking each other—deadlock—or in buffer overflow. Since the corpus only needed to be processed once, and all the files are of manageable size (less than 60 kilobytes) there is no particular need to adopt the most space- or time-efficient approach. The main justification for the approach we have chosen is its flexibility and the ease with which the various scripts could be tested. Since the scripts can each be called directly from the command-line, the order of calling can be tailored to the kind of test being performed. Also, files can be partially processed using some scripts, modified to exhibit some trait of interest, and immediately used as input to the script currently under evaluation. Had the processing been done as a series of subroutine calls (arguably the most efficient method possible), this would have been dramatically more difficult. Hence, though this approach is not optimally efficient, since it made debugging considerably easier, accuracy of the processing can be expected to be higher than it would have been had any of the other approaches we considered been selected. The coordination of the filters is not difficult, and will be described in section 2.9 below.

2.3 Locating and Removing Article Headers

One of the more pleasant traits of this corpus is that article headers have a very consistent format. Article headers have up to five lines. These are: first, 30 dashes (60 for the first article in a file, the one immediately following the file header). This line is followed by zero or one blank lines, then lines for the date the message was sent, from whom, and finally a line indicating the subject of the article. We have developed a script, described in detail below, which uses this format to replace article headers by unambiguous, easily-identifiable tokens.

For a segment of text to be considered as a candidate for an article header, we demand that it begin with a line consisting of exactly the correct number of dashes, as described above, and containing no other characters. We allow the subsequent optional blank line to contain any whitespace characters (here meaning either tab characters or space characters.) Further, the date, from, and subject lines are permitted to begin with any sequence of characters at all, whether whitespace or not; this is justified by the fact that, for example, a tilde (˜) precedes these lines in certain headers. The "date" line must have the word "date:", complete with colon, somewhere on it; the "d" may or may not be lowercase. It must be followed by a nonempty sequence of whitespace characters, then by at least three sequences of alphanumeric characters followed by whitespace characters or non-alphanumeric characters. Here, non-alphanumeric characters include such characters as periods, colons or slashes that are often used to separate the fields of a date. This ensures that a date will only match if the word “date” is followed by some text which could possibly be a date in some format (either words or numbers demarcated with some non-alphanumeric character). For the “from” fields, the word “from” may be in either case. The word “from” must be followed by a colon, a nonempty sequence of whitespace characters (and optional non-word characters), then two or more alphanumeric strings of any positive length, separated by non-alphanumeric characters. This allows matching of almost any name or e-mail address; even though very weak, this heuristic turned out to be overly strong in several cases where an article had originated with the moderator of the RISKS Forum, and so only the word “RISKS” followed the “from”. These cases were corrected manually wherever found—garbage data were added to make the field conform to expectations. The “subject” field must contain the word “subject:”, with colon, with the “s” of either case. No whitespace or non-alphanumeric characters need appear between the colon and the first alphanumeric character in the subject line—but some of either type may occur. One string of positive length of alphanumeric characters must occur after the colon—the article must have a subject. Since all RISKS files have tables of contents which use the information in the subject field, the assumption that at least one identifiable word will be present is entirely valid.

Once an article header has been identified, the script substitutes for it in the output an unambiguous, single-line token. Fifty consecutive @ signs was chosen, since a search of the entire corpus demonstrated that nothing like this token appears in the corpus. This substitution dramatically simplifies the deletion of article boundaries and the recording of their location, which is performed by a subsequent script, described in section 2.9. If the script detects a region of text that matches some, but not all, of the features of an article header, it leaves that region unchanged in the output text.

Finally, this script also deletes the file header. This is done because the structure of the file header is dramatically different from that of any article—it is a table of contents—

and this would clearly skew the results of an algorithm trying to use stylistic information to identify its boundaries. Further, since every file begins with a file header, it would be trivial for such an algorithm to identify the file header based on structural rather than stylistic cues. The fact that, by definition, this table of contents will be composed of phrases chosen by different authors would also introduce noise into the computation of an algorithm attempting to distinguish different authors on the basis of the style of the phrases they choose; i.e. file headers are not themselves contributions, as we have defined the term, and do not contain contributions of sufficient size to merit retaining them in the processed corpus. Hence, the presence of file headers would introduce unnecessary noise into the identification of subsequent contribution boundaries. So it was felt that they should be purged for all files.

Aside from the small number of errors caused by not correctly identifying “from” fields in article headers that were discussed above, and one or two cases in which fewer than 30 dashes were present, this script appears to be highly accurate. While no formal tests were carried out, since this script was used in conjunction with all other scripts that were formally tested, considerable indirect observation of its performance was made. We did not observe a single case in which text was incorrectly classified as an article or file header. Some cases were observed where the script generated a run-time warning regarding use of uninitialized values, but in these cases no effect on the output was observed. No case was observed where the script failed to identify and eliminate a file header. It seems fair to state that the script deleted file headers and replaced article headers very reliably, and thus that it does not introduce significant noise into the data.

2.4 Identification of Mis-Quoted “From”s

An unfortunate consequence of the fact that this corpus is composed of e-mail messages is that some e-mail programs are in the habit of placing quote marks in front of the word "from" when it appears as the first word in a line. Clearly, this is done because a "from" in this position could indicate that the following text was sent originally by some party other than the party who sent the e-mail being read; the e-mail program is merely reminding the reader of this possibility. By quote marks, here we refer to >, since this symbol is most often used to indicate quoted text in the e-mail domain.

Since it is our goal to find boundaries between authors' contributions, and hence we need to detect quotations—text supplied by other authors—this treatment of "from"s poses problems for us. Inasmuch as such lines are often, indeed usually, not part of quotations, and are actually commonly integral parts of text, before identifying quotations for further processing we must first ensure that lines of this type will only be treated as quotations if they are quotations. So, we have devised a script to remove quote marks from in front of certain "from"s. Such a "from" will be recognized if it is preceded by a > (and possibly whitespace, possibly between the "from" and the >.) To avoid deleting legitimately-placed quote marks, there must be no >'s at the beginning of the closest text-bearing line to the "from" in question. That is, there must be no > (with optional whitespace) at the beginning of the first line containing non-whitespace characters, either before or after the line containing the "from" in question. The case of the word "from" is not significant for this script. When the script recognizes a line containing a mis-quoted "from", it simply removes the quote marks at the beginning of the line.

As with the header identification script, no formal testing was done on this script. However, when the script was evaluated on an artificial test suite, no bugs were observed. Further, no errors have been observed in informal examination of real data treated by the script when it was used in conjunction with other scripts that were formally tested. It appears that this script does not introduce any noise into the data, and indeed that it helps to decrease the overall level of noise present in the data.

2.5 Locating Quotations

Of the many aspects of the e-mail domain which distinguish it from other literary forms, one of the most striking is the profusion and structure of quotations. For research aiming to train an algorithm to contrast authors using statistical measures of their style, little could introduce more confusion into the data than to attribute the work of one author wrongly to another. Viewed from this perspective it is plain that, without substantial revision, corpora from the e-mail domain would not be at all suitable for training or testing such an algorithm. Fortunately, to make it easy for humans to tell when information is being quoted, several formatting conventions have been adopted. By using these same formatting indicators, it has proven possible to develop an algorithm to tell quotations apart from original text with reasonably high probability. We use this algorithm to insert unambiguous markers around quoted material, as well as to remove the formatting indicators used to identify the quotations. In this section, we discuss this algorithm and describe the procedure used to verify its efficacy.

2.5.1 The Algorithm

The algorithm employed to detect quotations is in fact a set of rather complex heuristics, motivated both by a knowledge of this corpus in particular and methods of quoting generally employed on the Internet. While the most common method of quoting found in other domains, whereby a quotation mark is applied to the beginning and the end of a quoted passage, is used in this corpus as well, the most popular methods of indicating quotations in the e-mail domain can be divided into two groups. In the first group, some specific character, possibly with some whitespace added after it, is applied to the beginning of each line being quoted. Alternatively, some fixed amount of whitespace may be applied to each line of the text being quoted. Each of these techniques poses interesting problems, and so they will be discussed in turn.

We have found that three different characters—the greater-than symbol (>), colon, and the vertical bar (|)—are placed at the beginning of lines to indicate that those lines have been quoted. E-mailers that conform to this convention of indicating quotations often apply these symbols automatically to messages that are being forwarded or replied to. Furthermore, e-mail discussions often contain "threads"—series of messages in which two or more parties respond to each other's e-mails, often re-quoting substantial parts of previously-seen messages in order to preserve context. All this means that these symbols can appear, at least in theory, in arbitrarily complex combinations. In practice, we have seen instances in this corpus where three greater-than signs were used to indicate a quotation, or where two greater-than signs followed by a vertical bar were used. Hence, when we look for a string indicating that a quotation of this form has begun, we must match on a (possibly empty) sequence of whitespace characters, followed by a non-empty sequence of any one of our three characters followed by a possibly empty series of whitespace characters. This quotation-indicator will be terminated by a newline, or the first character outside the set of our three quote marks and the whitespace characters. Once such a sequence has been identified, the corresponding line of text is considered to be part of a quotation; we assume that our procedure for handling mis-quoted "from"s has already been run. When we first find such a sequence, we infer that a quotation has begun and store a begin-quotation token (described below) on the line preceding in our output array. We say that the quotation continues so long as the same pattern as was found at the beginning of the first line in the quotation matches the beginnings of subsequent lines that contain non-whitespace characters (we allow quotations to persist over blank lines). The only exception to this rule is for seemingly "quoted" lines immediately preceded and followed by non-blank, non-quoted lines. Such lines are not treated as quotations, since the quote marks may be fulfilling some content-related function. We have not observed a case in which a line of text was quoted in this way; authors appear to consistently isolate single lines of quoted material. Once we have deemed a quotation to have finished, we insert an end-quotation token on the next line of output, and continue processing our input. After the extent of a quotation has been determined, the prepended quote-indicator characters are stripped off every line, so that the text is formatted like a regular block of text. Also, to allow quoted paragraphs using the indented paragraph style (described in section 2.7) to be treated correctly, we have written the script in such a way as to include whitespace characters as part of the string to be removed from each line only when those whitespace characters are present on all non-blank lines of the quotation. This completes our treatment of this type of quotation.

Quotations that are indicated by prepending a certain amount of whitespace to each line are much harder to recognize. Some authors have a habit of beginning each line of their work exactly one space away from the left margin. While this is no doubt pleasant aesthetically, it means that we cannot treat text indented with one space character as quoted material. Fortunately, our testing (discussed in subsection 2.5.2 below) demonstrates this does not lead to a significant number of false positives. Other authors do not use a block style of paragraphing, preferring to indent the first line of each paragraph. This means that we cannot treat a single indented line of text as a quotation if it is followed by non-indented lines. If an indented line is preceded by non-indented text, then it may represent the run-over of an item in a list of points to a new line; thus we cannot treat such lines as quotations. Often, Internet URL's or FTP site names are placed on isolated lines and indented, in order that they may be located and cut and pasted with greater ease. Thus, we cannot treat as a quotation an isolated line of indented text beginning with a token such as “http”, “ftp”, “www”, “url”, or one of these tokens preceded by a character such as <, which is often used to set such information off. Originally, we had decided not to treat indented material appearing at the beginning of an article as a quotation, since some articles begin with titles, which are almost always indented. In our first round of testing, however, we discovered that this exception resulted in a very significant number of false negatives. It can also be argued that the structural cues provided by titles might make it artificially easy for an algorithm to pick out the beginning of a contribution. Believing it better to err on the side of caution, we have eliminated this exception. Therefore, any unit of text that begins with two or more space characters, or with a tab character, that is not covered by any of the above exemptions is treated as a quotation.

Indented quotations, once recognized, are treated in much the same way as quotations indicated with special line-initial characters. Effectively, we store a begin-quotation token on the line before the quotation, and an end-quotation token on the line following it, and attempt to strip all the indenting whitespace from the beginning of each line of the quotation. As with quotations indicated by quote marks, blank lines do not terminate quotations and are preserved within them.

Quotation mark-delimited quotations are not commonly used in the e-mail domain, and so we had initially planned to ignore them. However, after the results of our first round of testing indicated that an unacceptable number of large quotations were being missed because of this design decision, we decided to take them into account in our script's final version.

Quotations of this variety pose a number of problems conceptually as well as algorithmically. Chief among these is how to distinguish material inside quotation marks that was composed by another author, and hence is a "quotation" for our purposes, from material that we would not want to treat as a quotation. For example, when introducing terms, authors often use quotation marks; clearly these should not be treated as quotations—indeed, to do so would potentially deprive us of valuable insights into the style and word choice of an author. Other kinds of quotations, such as reports of conversations (which occur often in this corpus), form an integral part of the structure of the text; they might contribute phrases or clauses of sentences, for example, and so to excise these sorts of quotations would leave disjointed text that might not be amenable to further analysis. Thus, we have decided to identify only large segments of text appearing between quotation marks as quotations—text at the sentence or even paragraph level. Specifically, if we find a line which begins with either a " or ``, the characters most often used to commence quotations, and we find a quotation-terminator (", '' or ``), as the last non-whitespace character of the same or subsequent line of text, we treat all intervening text as a quotation. We also demand that no quotation marks appear in the text anywhere between the line-initial and line-final marks that we use as demarcators. We do not allow this type of quotation to persist over blank lines. While this is rather arbitrary and weak—we do not insist that the quotation marks match, for example—our testing leads us to believe that it at least finds most of the large blocks of text quoted in this way, which could introduce considerable noise into our data if left unidentified.

Our treatment of passages identified in this way is analogous to that given to passages using other quoting conventions. We insert begin-quotation and end-quotation tokens on the lines preceding and following the passage in question, and strip off the quotation marks we have identified.

We have selected a line of 50 consecutive “q”s as our begin-quotation token, and 50 consecutive “u”s as our end-quotation token. We insert blank lines after both these tokens in the output so that paragraphs are still identifiable. It is the responsibility of another filter, discussed in section 2.6, to further process the quotations once they are identified and made to resemble original text in their formatting.

2.5.2 Testing the Quotation Identification Algorithm

To test the quotation identification portion of this phase of the experiment, we first had to select a random subset of the files in our corpus. This we did by developing a simple script which used Perl's pseudorandom generation facilities to select and output a subset of its input lines, whose size is governed by a command-line argument. Then we sent this script a listing of our archive's filenames for its input, and used the subset of names that it returned as our test set. This is the same method we used to test our “signature” removal script, discussed in section 2.8.

As noted above, we ran through two iterations of testing. For both tests, we used ten files—approximately 5% of our entire corpus. The results of the first test are given in table 2.1.

We should first observe that the determination of what quotations actually were present in the test data was manual, and hence subjective and prone to inconsistency. We have tried to be very liberal in our interpretation of what constitutes a quotation; when an article body, for example, is composed almost exclusively of an extract from some publication, we have construed the extract to be a quotation even though the originator contributed virtually none of his or her own text. Since attributing quoted material to the author of an original work could introduce very considerable noise into our data, we felt that identifying only 76.8% of the quotations present was unacceptably poor performance. An examination of the quotations that were not recognized led us to treat indented material at the start of articles as quotations, as mentioned above. Also, we added the capability of handling some instances of the use of traditional quoting methods to our script. After implementing these features, we tested our algorithm once again on another ten files, and came up with the results given in table 2.2.

As in the previous test, we tried to err on the side of caution when assessing what is a quotation. Despite this we achieved over 95% accuracy in identifying quotations. Further, the standard deviation of our missed quotations was cut in half, providing some indication that there were no large clusters of missed quotations. We are confident that it would be extremely difficult to reduce the percent of missed quotations further, since most of the missed quotations were defined by context—e.g., an article body would contain an introduction, then a forwarded message, without using format to indicate the presence of a quotation. Another example of a type of quotation that would be very difficult to spot is a quotation demarcated by square brackets. We suspect that, over the entire e-mail domain, these would be quite infrequent; however, since this is the style the moderator

File Name	Num. of False Positives	Num. of Unidentified Quotations	Num. of Correctly Identified Quotations	Total Num. of Quotations
18.08	3	0	6	6
18.56	1	1	2	3
18.87	1	1	9	10
18.96	0	3	3	6
19.10	0	5	6	11
19.45	3	1	4	5
19.69	1	1	7	8
19.70	0	2	6	8
19.95	0	1	4	5
20.14	2	1	6	7
Total	11	16	53	69
Mean	1.10	1.60	5.30	6.90
Percent	17.18	23.20	76.80	100
Std. Deviation	1.20	1.43	2.06	2.42

Table 2.1: Results obtained by running the first version of the quotation identification script on pseudorandomly-chosen files from our corpus.

The percent entry in the false positives column refers to the percent of marked entities that were not quotations.

File Name	Num. of False Positives	Num. of Unidentified Quotations	Num. of Correctly Identified Quotations	Total Num. of Quotations
18.06	5	2	10	12
18.22	2	0	14	14
18.33	1	0	4	4
18.37	3	0	8	8
18.71	2	0	5	5
19.30	2	1	8	9
19.43	4	1	9	10
19.61	3	0	5	5
19.76	0	0	10	10
20.10	0	0	8	8
Total	22	4	81	85
Mean	2.20	0.40	8.10	8.50
Percent	21.35	4.70	95.30	100
Std. Deviation	1.62	0.70	2.96	3.21

Table 2.2: Results obtained by running the final version of the quotation identification script on pseudorandomly-chosen files from our corpus.

As in table 2.1, the percent entry in the false positives column refers to the percent of marked entities that were not quotations.

of the `comp.risks` newsgroup uses to interject his own comments, its frequency in this corpus is not by any means insignificant. Normally, the moderator also uses indentation, so that our tests for indentation succeed at identifying these quotations. However, in the cases where this is not done we cannot detect this type of quotation. Since square-bracketed material does not often denote the presence of quotations, and indeed could form a significant part of an author's style, we believe that to include this cue would not only dramatically increase the proportion of false positives, but would do so in a systematic way that would bias the resulting data. Finally, it is interesting to observe that, though the number of false positives doubled between tests, the proportion of false positives to correct identifications only increased marginally. We feel very confident that to further increase our recall, however, we would begin to drastically increase the rate of mis-identification; thus, we have been content with the accuracy we achieved on this test.

2.5.3 Nested Quotations

Before moving on to describe what we have done with the quotations we have identified, we must discuss the matter of "nested quotations." By a nested quotation, we mean a quotation that appears inside another quotation. While this is less common in our corpus than might be expected (possibly because our corpus is tightly moderated and hence substantial re-quoting is kept to a minimum), nested quotations nonetheless occur with sufficient frequency to warrant consideration. The algorithm we have presented above is simply designed to detect a single layer of quotations; it reads through the document only once, and considers each line in sequence; hence, it is not possible for the algorithm to distinguish a nested quotation from a non-nested quotation, since, at least in the case of quotations indicated by formatting, both can have the same formatting indicators. Fortunately, if an outer quotation precedes an inner quotation and the algorithm successfully detects the outer quotation, it simply treats the nested quotation as regular text—which means that the nested quotation maintains its integrity. That is, if the nested quotation is indicated by having each line preceded by one more > symbol than occurs for each line of the outer quotation, after processing, every line of the inner quotation is still preceded by one > sign. Thus, if the algorithm is run over the text produced by its previous run, the inner quotation can be successfully identified and marked. If no part of the outer quotation precedes any part of the inner quotation, the algorithm simply treats them as two separate quotations, marking them separately in the text. For reasons that will become clear when we discuss what we have done with the quotations we have found (section 2.6), this effect suits our purposes well. In practice, we have found this technique extremely effective. Moreover, we have found no case in which it is necessary to apply the algorithm more than twice to our corpus. Thus, we are able to accurately detect nested quotations by simply circulating our corpus through the algorithm twice.

2.6 Moving the Quotations

Once we have reformatted quotations to resemble original text, and demarcated them in such a way that they can be located and their extent determined unambiguously, the question immediately arises as to what should be done with them. At least three possibilities suggest themselves: we could excise all quotations from the corpus; we could leave them in place, counting their boundaries as contribution boundaries; or we could treat them as contributions in their own right, but move them to new locations so that the original text of which they were a part is made to be contiguous.

None of these proposals is free from disadvantages. To simply eliminate quotations entirely would substantially diminish the size of our corpus. There is also an argument which hypothesizes that, should our methods succeed, it will be due to the fact that articles have structure, and that our algorithm has learned not to distinguish style but rather to distinguish article conclusions from article introductions. Since one would not expect quoted material to obey the same structural constraints as entire articles, testing the algorithm on quotations might prove to be highly instructive. But simply leaving the quotations in place and treating them as contributions will dramatically increase the frequency of authorship changes in our corpus. We have already admitted that our task of training an algorithm using the data we plan to generate will be very challenging. It therefore seems reasonable to try, wherever possible, to preserve auctorial contributions—that is, not to break up individual contributions unless absolutely necessary. This follows from the fact that, the more authorship changes we introduce, the less easy it will be to train our algorithm on the resultant data. Thus, we are persuaded that the third proposal offers the most promise.

This option poses two challenges. The first, where to put the quotations once they are extracted from the text, we have solved very simply. Largely to make the task of examining our algorithm's performance on quotation-contributions easier, we have decided simply to store the quotations at the beginning of the file in which they were found. Since the order in which the quotations are recorded does not seem to be significant, for ease of implementation we store them in reverse order from their occurrence in the original document. Internally, of course, the structure of each quotation is completely preserved.

The second problem posed by our decision to move quotations but treat them as individual contributions is far more subtle. One does not have to have vast experience with the Internet to realize that, in most threads, one tends to have a quoted paragraph, followed by a response; it is highly uncommon that more than one paragraph is quoted and then responded to. In itself, this fact will tend to skew our data—our algorithm may do very well simply by guessing that an authorship boundary occurs after each paragraph. Worse, quotations can be extremely short—one- or two-line quotations are not uncommon at all. Nor do quotations have to be complete sentences: often phrases or relative clauses will be quoted as providing sufficient context for a response.

Our method for dealing with these facts is both straightforward and arbitrary. We demand that, in order to remain part of our corpus, each quotation have at least four text-bearing lines, regardless of how many paragraphs it contains. On the one hand this does, in some measure, ensure that a quotation contains enough text for a meaningful amount of stylistic information to be retained within it. On the other, this increases the likelihood that a quoted passage has more than one paragraph—thus lessening the amount of skew introduced into our data by the presence and concentration of quotations.

Once we had made these decisions, the implementation was trivial. The script developed to detect and move our quotations simply looks for the begin-quotation token and adds all subsequent data to the beginning of the array storing the output, until the corresponding end-quotation token is found. Care is taken within the script to allow for nested quotations, and to store them sequentially—inner quotations are extracted from outer quotations and stored before them in the output data. This script also removes "empty quotations"—quotations containing only lines of whitespace characters. These occur as noise occasionally produced by our quotation-detecting procedure. Finally, this script removes all end-quotation tokens, relying on article boundary tokens and begin-quotation tokens to serve as authorship demarcators in the output. The job of removing quotations with fewer than four non-blank lines is left to the final step of processing, discussed in section 2.9.

Since this procedure is quite straightforward, an artificial test suite was considered sufficient to evaluate its accuracy. In this evaluation, and in examination coinciding with the formal evaluation of our signature-removing procedure, the script was not observed to make any errors.

2.7 Reformatting Paragraphs

Since we have decided that, for the purposes of this experiment, parts—the units from which our stylistic statistics are to be derived—will correspond to paragraphs, we can make computing these statistics much simpler by ensuring that all paragraphs in our corpus are in a standard format. Further, as described in detail below our signature removal procedure operates only on paragraphs; this procedure will be made more accurate and reliable the more accurate and reliable our procedure for identifying paragraphs. Thus, it behooves us to standardize the format of paragraphs in our corpus as far as is possible.

The most common, and the most tractable, paragraph format to be found in the e-mail domain uses blank lines (or more generally lines composed only of whitespace characters) to mark the boundaries of paragraphs. This is the format to which we would like to reduce all paragraphs, since it is very simple and the boundaries between paragraphs are totally unambiguous. Some authors prefer to adhere more to the traditional style of indenting the first line of their paragraphs, and so leave some whitespace in front of this line, but nevertheless still include a blank line between each paragraph. While these indentations could potentially confuse analysis that looks for writing in point form, because the paragraph breaks are unambiguous we have made no attempt to alter this formatting. Other authors, however, rely strictly on indentation of the first line to indicate the extent of their paragraphs. Since such indentations are very difficult to distinguish from indentations used to set off the items in a text written in point form, we must reformat this type of paragraph.

The most salient feature of the procedure we have adopted for reformatting this type of paragraph is its conservatism. As much as the potential for missing paragraph boundaries could introduce noise into our data, to introduce paragraph boundaries that are not present in the original corpus would do even more harm, since they could destroy the syntactic integrity of the original text and hence invalidate the statistics computed from that text. For this reason we feel justified in demanding that articles to be reformatted conform to strict conditions. Since we run this procedure on our corpus after marking and moving quotations, and it would appear that the decision on what paragraph format to use depends on an individual author and hence is likely to pervade all the text of a contribution, we feel that it is reasonable to decide that once we identify a contribution as having paragraphs that need reformatting, the reformatting should be applied throughout that contribution. To identify a contribution as requiring reformatting, we demand that its first text-bearing line contain an indentation, be followed by at least two non-indented text-bearing lines, and that this pattern be repeated at least once with no intervening blank lines. Since we require the first text-bearing line to contain an indentation, it is highly unlikely that we will mistake a contribution containing material written in point form for one needing reformatting. Our requirement that each paragraph be at least three lines long in total, and that there be at least two such paragraphs at the start of the contribution, gives us further assurances in this regard. Once we identify a contribution as requiring reformatting, we simply delete all indentations and place a blank line in the output before the line which was originally indented.

We have tested the script which implements this procedure on some artificial test data, and have found its performance to be satisfactory. Since this paragraphing style is relatively uncommon in our corpus, we have only observed two instances where this procedure should be applied; in both cases it was triggered and was successful. We have no other means of verifying that our implementation is free of defects or that our conditions for triggering its application are sufficiently—or overly—strong. Nonetheless, we are confident that we have removed at least some noise from our corpus by developing and implementing this procedure.

2.8 Removing Signatures

There is little doubt that, in some sense at least, style and an author's signature are intimately related. That one person might choose a signature containing his employer's name, his job title, his e-mail address and name while another chooses one containing only his name and a third chooses not to use any signature at all may say a great deal about the type of writing these people are likely to produce. But, however much information there may be in signatures, we cannot use them for our research. It is, after all, not hard to distinguish a signature from a non-signature with high probability—this section demonstrates a method that, with very slight modifications, would do precisely that. If we allowed signatures to persist in our corpus, we need do nothing more than use this very method to search for signatures, then predict authorship boundaries to occur immediately afterwards in order to achieve a high authorship boundary recovery rate. Indeed, such a procedure might be useful as a baseline comparison to the performance of the neural nets we will discuss in chapter 5. Since we wish to investigate whether more subtle evidences of style can allow us to achieve a good boundary recovery rate, we must eliminate signatures from our corpus to as great an extent as possible.

One assumption underlying our treatment of this subject that we must immediately state is that signatures can only occur as the last identifiable paragraph of text in an article body. This assumption is by no means always valid; postscripts and post-postscripts usually occur after signatures, for example. Also, the moderator of this list very often inserts comments that pertain to the article immediately after its last paragraph. Finally, some authors do not have their signatures as paragraphs, but prefer to put their names at the end of the last line of text in their articles.

Fortunately, as mentioned above, we have been quite successful in identifying the moderator's comments as quotations; when identified, they are then moved out of the article. In these cases then, the author's signature, if it exists, will indeed occur as the last identifiable paragraph in the contribution. We have seen no obvious way of dealing with postscripts, or with signatures that are part of the last line of text. Fortunately, as the testing outlined below shows, these cases are not common.

The heuristics for detection of signatures are quite complex, and motivated by general knowledge of the Internet and secondarily by knowledge of this corpus in particular. As with the quotation-finding procedure, we were forced to proceed in two steps, but the modifications for the second phase were not as extensive as in the quotation-finding case.

2.8.1 Detailed Explanation of the Heuristics

Our fundamental contention is that the purpose of a signature is to convey certain information, so that to recognize a signature it suffices to recognize these indicative pieces of information. There are seven types of information that signatures usually contain: the author's name, e-mail address, phone number, URL, postal address, job title (possibly including company name), and erudite or witty quotations which authors use to personalize their signatures and make them more interesting. We choose to define paragraphs that contain certain combinations of these features as signatures, positing that most random paragraphs have a low probability of containing such combinations so that we falsely identify as few paragraphs as possible. We discuss these combinations, and how we detect each unit of information, in the remainder of this subsection.

The most obvious type of signature is one containing a single name. We also count as signatures those paragraphs containing an identifiable name, along with any one of an e-mail address, a URL, or a phone number. For this type of signature, we demand that the identified name be separated from surrounding text in some way. This separation may be a single tab character, two or more whitespace characters, a comma, or by whitespace followed by a less-than symbol (often used to enclose e-mail addresses or URL's.) Further, we count two-line paragraphs whose second line contains a name and whose first contains one or two words as a signature—some authors, such as the author of this thesis, preferring to close their articles with a salutation such as "cheers," followed by their name. As noted below in our description of our procedure to identify names, these are not easy to identify; thus, we count as a signature any paragraph containing two of an e-mail address, a URL, or a phone number. Finally, some authors simply include their e-mail address as a signature, and so we allow a single e-mail address also to constitute a signature.

Unlike the other features, postal addresses, titles, and profound quotations are not at all easy to identify. Recognizing that all these features tend to contain a high proportion of words beginning with uppercase characters, we have used the heuristic of treating as signatures paragraphs containing high proportions of words beginning with uppercase characters, regardless whether they contain any identifiable names, e-mail addresses, URL's, or phone numbers. For this purpose, we count as words only text beginning with some alphabetic character that has no alphabetic character immediately to its left; numbers, for example, are not considered words under this simplistic definition. At first we planned to demand that 75% of identifiable words in a paragraph be uppercase before treating that paragraph as a signature. Our first round of testing showed that this allowed a considerable number of legitimate signatures to pass through unidentified; thus, in our final version we lowered the threshold to 60%.

Having discussed the combinations of types of information that we use to identify signatures, we must turn to the nontrivial task of describing how we have proposed to identify those features—i.e. names, phone numbers, e-mail addresses and URL's—that we considered to be identifiable. Of these four, names are probably the hardest to detect. In general, we define a name to be a string of one to three words consisting of alphanumeric characters, and demarcated by whitespace, periods, or dashes. We include periods to allow for initials, dashes for hyphenated last names. We do not insist that the first alphanumeric characters in words be uppercase, since, while a common convention in formal writing, it is not altogether pervasive in e-mail signatures. This will not catch all names—we would miss, for example, John E. Smith Ph.D., since this contains more than three words. However, since the definition is already very broad, and in practice we have found it fairly comprehensive, we have chosen to include a maximum of three words in a name. If a name is inside a paragraph—that is, it is included along with information such as an e-mail address—we demand that the words begin with uppercase characters. In this case, these signatures have usually been made up in advance and are included automatically with each article the author sends, so it is likely that the uppercase convention will be observed. A name in isolation (or in a two-line signature with a salutation) will often be typed as needed, which explains why the uppercase convention is commonly ignored.

Contributors to our corpus reside in many nations, and this is reflected in the variety of phone numbers to be found in their signatures. We have thus had to adopt a very liberal definition of what constitutes a phone number: specifically, a phone number is either composed of two strings of digits, separated by a space or a dash, whose first constituent has length 2 or 3 and whose second constituent has length 3 or 4, or else is a series of three or more strings of digits, each of length 2 or 3, separated by periods. While this definition encompasses some postal addresses—and, more significantly, also overlaps some styles for citing legal documents such as sections of statutes—it has nonetheless served us well enough in practice.

E-mail addresses are rather more standardized than phone numbers, and hence are more tractable. We define an e-mail address to be a nonempty string of alphanumeric characters followed by an `@` sign, followed by a nonempty series of strings of alphanumeric characters, demarcated by periods, and ending with two or three alphabetic characters. We also permit dashes and underscores to be parts of the "alphanumeric" strings referred to in the preceding sentence. This pattern matches every e-mail address we have tested it on; we have not observed it to produce a false positive in any test.

URL recognition is somewhat more complex than the recognition of e-mail addresses. First, we demand that our URL's begin with one of "http://", "www", "ftp", or, in upper- or lowercase, "url". These flags provide a strong indication that a URL is present, but we also demand that a domain name be present—that is, that a nonempty series of alphanumeric strings, demarcated by periods, and terminated by a string of two or three alphabetic characters, be present. While it may be argued that this is more strict than is necessary, we want to make certain that we do not confuse e-mail addresses with URL's;

this is why we require one of the identifying flags to be present as well as the domain name.

2.8.2 Testing

Now that we have completed our lengthy discussion of what combinations of features we consider to indicate a signature, and of how we have detected those features, we will proceed to a description of our testing. As in our testing of our quotation identification procedure, we selected ten files from our corpus using the pseudorandom number generation facilities provided by Perl. The results we observed in our first test are presented in table 2.3.

While this table demonstrates that the first draft of our signature-removing procedure is reasonably effective, we decided to implement several modifications in hopes of increasing its accuracy further. First, we fine-tuned our procedure for recognizing signatures composed of names and e-mail addresses. At first we did not permit names to be followed by only one whitespace character. The results in table 2.3 convinced us to recognize names that were followed by a tab character, as well as those followed by some whitespace character and a less-than symbol. We also expanded our definition of what constitutes a phone number to include period-separated numbers—as discussed above. It was at this stage that we decided to lower the percentage of uppercase words from 75% to 60% which a paragraph must contain in order to be counted as a signature. Finally, we observed several instances where single-paragraph articles were treated as signatures and thus had their entire text excised. In an attempt to avoid this, if the text of a single-paragraph article occupies four or more lines, we only examine it for a high proportion of upper-case characters; we do not subject it to e-mail address, name, URL or phone-number detection. Since these changes are not fundamental, we decided to verify their effectiveness with only a test of five randomly-chosen files. The results of this test are shown in table 2.4.

File Name	Num. of False Positives	Num. of Unidentified Signatures	Num. of Correctly Identified Signatures	Total Num. of Signatures
18.08	2	1	14	15
18.21	3	2	7	9
18.41	1	3	7	10
18.48	1	2	6	8
18.74	4	0	9	9
19.48	4	2	8	10
19.83	1	4	8	12
19.87	2	1	7	8
19.92	0	2	11	13
20.13	1	0	9	9
Total	19	17	86	103
Mean	1.90	1.70	8.60	10.30
Percent	15.570	16.50	83.50	100
Std. deviation	1.37	1.25	2.50	2.31

Table 2.3: Results obtained by running the first version of the signature removal script on pseudorandomly-chosen files from our corpus.

As in table 2.1, the percent entry in the false positives column refers to the percent of deleted entities that were not signatures.

File Name	Num. of False Positives	Num. of Unidentified Signatures	Num. of Correctly Identified Signatures	Total Num. of Signatures
18.15	0	1	10	11
18.19	1	1	10	11
18.41	1	2	8	10
19.28	2	0	6	6
19.63	6	1	6	7
Total	10	5	40	45
Mean	2	1	8	9
Percent	18.18	11.1	88.9	100
Std. deviation	2.35	0.71	2.00	2.45

Table 2.4: Results obtained by running the final version of the signature removal script on pseudorandomly-chosen files from our corpus.

As in table 2.3, the percent entry in the false positives column refers to the percent of deleted entities that were not signatures.

Due to the smallness of the sample size for the tests described in table 2.4, we must take care not to attach too great a significance to them. Also, one of the files in the second sample was the same as in the first; this may have biased the result. However, these results would seem to indicate that our changes to the signature-removing procedure were effectual, if not quite to the degree we might have hoped. It is not obvious to us how we might improve the accuracy of our signature removal without dramatically increasing the incidence of false positives. We have trouble with recognizing signatures exactly for the reasons mentioned at the beginning of this subsection: they are highly individual and therefore are incredibly varied. The humorous or learned quotations which many authors include turn out to be quite as hard to detect as we had feared. Formats of signatures vary wildly: some authors are very plain, but others ornament their signatures with ASCII graphics or align portions in columnar formats. As we stated before, we have not even attempted to handle signatures appended to the end of the final paragraph of the contribution. All this shows that achieving greater accuracy in signature detection would be more arduous than the benefits derived therefrom would appear to justify. Hence, we have been content with removing just under 90% of the signatures, and accept an almost 20% rate of false positives.

2.8.3 Removing File Trailers

The final, trivial task we have accorded to our signature-removing script is to remove what might be considered as the "signature" of the corpus itself—the final article in every file, which contains useful information such as the location of the `comp.risks` archive, how to subscribe to its mailing list, who the moderator is, etc. It seemed clear to us that the constant repetition of this data in the training set might bias the algorithm. Indeed, we could not see a reason why this repeated article should be preserved: we still have a vast amount of text at our disposal, so its presence adds nothing. Since it would certainly introduce a bias into our training set, it seemed to us imperative that this article be eliminated in all files.

There are other articles which are repeated from time to time in the corpus: for example, in roughly one out of ten files, an article is included that reminds readers of the existence of the Privacy Forum and the Computer Privacy Digest, two fora related in content to `comp.risks`. While even this repetition will likely bias our training to some degree, the bias introduced will be far less than that introduced by an article repeated in every file. Further, there is an argument to be made that it would be interesting to examine the performance of our trained algorithm when given data that it has seen before in a context it has not observed previously; these repeated articles provide us with a good means of examining this facet of our trained algorithm's performance.

For these reasons, we have decided that the file trailer would be the only article we systematically eliminate from the corpus. To achieve this, we simply suppress in our output all text following the last contribution boundary found in the input file. As discussed in the next section, this part of our procedure was perfectly accurate with the exception of one file in the corpus, where the error was obvious and was corrected manually.

2.9 Recording the Article Boundaries

Now that we are able to identify all the boundaries between contributions, we are finally ready to go through our corpus and remove the tokens we have used to demarcate contribution boundaries, and record where those boundaries are. Simultaneously with this activity, we do some final cleaning up on our corpus.

For the statistics-gathering phase of our experiment at least, we have decided to maintain the division of our corpus into files. To facilitate access to the records of the locations of contribution boundaries, we have decided to centralize these records in one file. Thus, it makes sense for the script that determines the locations of the contribution boundaries to be responsible for coordinating the activities of our other scripts, since this script must remain in operation throughout the processing of the entire corpus in order to write the contribution boundary information to a single file. This motivates our incorporation of the running of all our other procedures into this script.

We elected to use a reasonably simple format for the contribution boundary database. Each line of the database file corresponds to a file of the corpus, and begins with that file's name. It then consists of a series of pairs of positive integers, each surrounded by parentheses and whitespace. For the i-th pair on some line, the first element of the pair represents the line number on which contribution i begins in the file named at the beginning of the line; the second element of the pair is the paragraph number of the first paragraph of contribution i. Line numbers, naturally, refer to the sequence of lines of a file; paragraphs are similarly numbered. In both cases, the numbering begins at 1. We also maintain a record of where the last contribution ends—that is, if there are n contributions in a file, we store the line and paragraph number where the $n + 1$-st contribution would begin. We have chosen to store both line and paragraph information partly to make statistical examination of the lengths of paragraphs and contributions easier, and partly because it was not clear at this stage which type of data we would want to use when training our algorithm or examining its results.

As well as removing the quotation and article boundary tokens left in the text form previous processing, this script is also responsible for removing noise such as consecutive blank lines, empty articles, and quotations that are less than four lines long; for an explanation of this last feature, see section 2.5. The script has been designed to process a series of corpus files, given either as command-line arguments separated by spaces, or as standard input separated by newlines. This not only facilitates processing the entire corpus, but permits one to test this script and the collection of procedures it relies upon on specific files of interest.

The functioning of this script was tested on both artificially-generated test suites and on small amounts of real data from the corpus. Since its functions are, in the main, fairly straightforward, it was not felt that a formal test, such as that carried out for the quotation-identifying or signature-removing scripts, was in order. We have not observed any bugs in this script.

2.10 Results of Processing the Corpus

We had remarkably few problems in running the series of scripts discussed in the preceding sections on the entire corpus. Of the 221 files processed, we observed just six warning messages: five were generated by our header removal script, and appeared to cause no problems (as noted in section 2.3, this is not uncommon for this script). One warning was generated by our quotation-moving script, which also does not appear to have caused any serious degradation of our data. One file, `20.06`, was seriously damaged by our signature removal script: it is the only file in the corpus which does not have the characteristic file trailer described above, and so its last article was eliminated. Unfortunately, the reason for its lack of a trailer was that there was only one article in this file, so it seems it was not considered necessary to append a trailer. We solved this problem manually by appending a trailer and running our file processing script, as described in section 2.9, on this file alone.

The above result adds yet more force to our assertion that the scripts we have developed in this phase of the experiment have few bugs, and have predictable effects on the corpus. In this phase, we have attempted to reduce the amount of noise in our data—through finding and isolating quotations from original articles, removing features such as signatures and article headers that correlate strongly with contribution boundaries, and reformatting paragraphs that are hard to detect. While some of our processing—particularly signature deletion and quotation identification—are not perfectly accurate, we are confident that we have achieved a degree of accuracy sufficient to allow us to state that, if our result is negative, it will not be because our data were too noisy.

After completing the task of initially processing the corpus, we observe that we have just over 750,000 words remaining, according to the standard Unix command `wc`, and about 4.5 megabytes of text. Both measures show that our initial processing has reduced our corpus to about three-quarters its original size. Clearly we still have a tremendous amount of data with which to work.

Chapter 3

Statistics Computed in this Experiment

3.1 Introduction

In this section, we discuss several categories of statistics that have been used in literature on authorship attribution, stylistic analysis, genre detection and information retrieval. Insofar as is possible, we discuss the motivations for and definition of these statistics, and then proceed to investigate the applicability of each statistic to our experiment. Recalling that one of the goals of this experiment is to determine which statistics work well on samples as small as those present in our corpus, we will compute values for all statistics except where there are very good reasons not to do so. We pay particular attention to these cases in this section.

Throughout this section, let N denote the total number of identifiable textual items, often referred to as tokens in the stylistic literature, to be found in a sample. Let V be the number of orthographically-different tokens in a sample; in the literature, V is often referred to as the number of "types". The concept of "type" is a general one: though the words "actual", "actuality", and "actually" all have the same root, they are all considered to be of different types. For a given sample, let V_i be the number of types that appear exactly i times; i.e. V_1 is the number of types (and therefore tokens) that appear exactly once in a sample, also known as "hapax legomena." "hapax dislegomena" refers to V_2; both terms are commonly-used in the literature.

3.2 Entropy and Juola's Measure

The concept of entropy derives from thermodynamics, where it is used to characterize the amount of order within a system. In thermodynamics, the higher the temperature within a system, the more movement will be observed among its component particles and thus the higher the system's level of chaos and the higher its entropy. Following the lead of the information-theoretic community, investigators of style have borrowed the concept in an attempt to measure the amount of structure in an author's writing. They use the formula

$$H = -\sum_i P_i \log P_i \tag{3.1}$$

where they define P_i to be the probability of the appearance of the i-th word type—i.e. P_i is the total number of occurrences of tokens in types appearing i times, divided by the number N of tokens in a text;

$$P_i \stackrel{\text{def}}{=} \frac{iV_i}{N}. \tag{3.2}$$

In theory then, the more structured—or at least the more homogeneous—an author's writing, the lower its entropy; the more disordered—or varied—the higher the observed entropy. It is then postulated that an author's oeuvre displays a characteristic amount of structure, and thus a characteristic entropy.

A slight modification to the entropy formula has been proposed, called the "diversity" of a text; this is

$$H \overset{\text{def}}{=} -\sum_i P_i \frac{\log P_i}{\log N}. \tag{3.3}$$

This measure returns the value one when all the words in a text are different, and zero when they are all the same (the text is completely uniform). Holmes [7] reports that it has been shown that this measure has little theoretical justification, however. Further, the entropy measure itself may provide valuable information. Since we will need to calculate V_i for some of our other statistics, there seems no reason not to compute the entropy, even though it will behoove us carefully to watch its impact on our networks if they are trained using it. Since the formula for the diversity adds a seemingly redundant factor to the formula for the entropy, it would appear that we would gain nothing by computing the diversity.

While previous research has concentrated on entropy at a lexical level, recent work by Juola takes the use of entropy to the level of characters. In [11], Juola explains the process by which his method can be applied to authorship attribution studies. For a sample of text C characters long, Juola selects a "window-size" parameter c such that $0 < c < C$. A "window" is composed of c consecutive characters from the sample. Suppose such a window begins at index i (i.e., at the i-th character),

$$1 \leq i \leq C - c.$$

Juola then calculates the length of the longest sequence of characters, beginning at index $c+i+1$, that is identical to some sequence of characters found within the window. Letting this quantity equal L_i, Juola's statistic is then defined as

$$\hat{L} \overset{\text{def}}{=} \frac{\sum_{i=1}^{C-c} L_i}{C - c}. \tag{3.4}$$

The method can be shown to converge to the information-theoretic entropy as $c \to \infty$; [11]. While in [10] Juola has worked with samples as small as 100 characters in comparing works in different languages, his work on authorship attribution used windows of 500, 1000 and 2000 characters. His cross-language work appeared to show that 250 was an optimal length; each of his latter measures worked well in authorship attribution [11].

It is true that most writers on the statistical treatment of literary questions view the lexeme as the ideal unit for computation, since lexemes are in some sense the basic unit of thought and thus most aspects of an author's idiolect can be expected to manifest themselves at the lexical level. Thus, most scholars in this community likely view Juola's work with some skepticism. Nonetheless, Juola's apparent success with small samples shows that his method might provide us valuable information—information that the more traditional stylostatistics, with their reliance on large corpora, might not be able to provide. So it makes eminent sense for us to utilize Juola's measure.

Given the smallness of our samples, a window length of 250 characters is too large to permit the achievement of a meaningful number of measurements on a significant proportion of our corpus. Therefore, we will use a window size of at least 50 characters, or $\frac{1}{4}$ of our text, up to a maximum of 250 characters. For paragraphs containing less than 200 characters—so that $\frac{C}{4} < 50$—we will use an estimate of the expected entropy of English text.

Seemingly a very basic statistic, the actual value of the expected entropy of English is much debated. Teahan [30] cites studies that, depending on the alphabet considered and the method of estimation, have plausibly determined an upper bound on this measure to be anywhere from 1.3 (Shannon) to 5 (Church). Both Teahan and Manning and Schütze [18] cite a very extensive study in which Brown et al determine an upper bound to be 1.75. Since the majority of interest in entropy comes from the data compression community, which uses it to estimate the ideal compression rate of English text, only upper bounds seem to have been calculated. We must do more research on this question, but Brown's estimate will likely prove our best choice both because of the amount of text on which it is based and because of the fact that the full range of symbols (not just alphabetic characters, as in Shannon's study, for example) was used in its computation.

3.3 Word-Length Statistics

Holmes [7] cites several studies that used word-length distributions. These authors simply recorded the proportion of words in a sample of text of length i for all lengths greater-than zero. These scholars viewed words as being drawn at random from an author's frequency distribution; thus, the comparatively simple statistical tests they performed with the word-length frequencies were justified.

Holmes [7] cites subsequent work, chiefly by Smith, which finds that word-length distributions, far from characterizing the work of any particular author, depend far more on the genre in which an author is writing or even the point in an author's career at which writing occurs. He also found that, within certain genres, word-length distributions seem not to differ significantly between authors.

Smith's findings regarding word-length distributions changing over time (both in general and with respect to individual authors) are clearly irrelevant to our experiment, since newsgroup articles tend to be written over short periods of time, and our entire corpus spans a period of only three years. Further, we have no interest in matching one post by an author with a subsequent post. Finally, all our samples are from the same genre. On the other hand, our samples will on average contain around fifty words; this leads us to be very concerned that bias may be introduced into our data through the smallness of our samples.

Nonetheless, since this statistic is trivial to compute, we will compute it. We will compute word-length frequencies for words of lengths $i, 1 \leq i \leq 15$. We choose fifteen as an upper bound since the MRC2 dictionary [35] documentation shows that less than 0.6%

of words are longer than fifteen characters. We will include words with more than fifteen characters in the count of fifteen-character words; this of course biases the frequency data, however, since use of such long words can be expected to mark an author's style, this bias should not be detrimental. We recognize in advance that it is most unlikely that this statistic will prove of much value.

We will also compute the average word length for each sample. Since this is a summary statistic, it can be expected to exhibit less variance than do word-length frequencies. Indeed, it would seem unwise to use both in the same test, so we will take care to conduct our tests using one or the other of these measures.

3.4 Syllable Distributions

A tremendous amount of research has gone into examining whether authors display characteristic patterns of syllable usage. Holmes [7] cites work that involved the calculation of the average number of syllables per word, the relative frequencies of i-syllable words, and the distribution of gaps between i-syllable words. Various genres—both poetry and prose—can be identified by these characteristics. It has been shown that frequency distributions discriminate languages more than specific authors. Other work has gone into developing distributions that characterize an author—that is, which can predict the frequency of i-syllable words that will be used in a text by that author. Other work demonstrates that syllable counts from consecutive words are to a large extent independent, but may depend on genre.

That many scholars hold that valuable information can be extracted from an author's syllable usage is plain. Much of the work, particularly in modeling syllable distributions, is rather immature; it has not been tested extensively enough—nor accepted widely enough—to justify our including it in this experiment. The work investigating gaps between words with certain syllable counts lacks a good cognitive underpinning—it is not easy to imagine how an author could display a consistent pattern with regard to placing certain gaps between words with certain syllable counts. The work on relative syllable frequencies, as well as that using syllable averages, does seem very relevant. It is true that syllables and word-length will correlate very strongly, so that we need to ensure our tests are not biased by too much influence from these statistics. Nonetheless, it will be worthwhile to compute frequencies of all i-syllable words in a given sample, for $1 \leq i \leq 6$. We will count words with six or more syllables in one category, since the MRC2 dictionary documentation states that less than 0.2% of words contain seven or more syllables. As in the case of the letters/word statistic discussed in section 3.3 above, this will introduce a bias in favour of long words; since use of such polysyllabic words would clearly mark an author's style, we do not view this as a serious drawback. We will also compute the average syllables/word for each sample.

The MRC2 database will be invaluable in helping us to compute these statistics, providing us with the source of syllable counts for most of the words in our corpus. There will, of course, be many words in our corpus which do not appear—or for which syllable counts are not given—in the MRC2 database; the database is, after all, largely based on lexical information from the 1960's and 1970's. We have therefore examined the mean ratio of letters/syllable for all words, and for words of each possible length, for which records exist in the MRC2 database. We present the results of this investigation in table 3.1.

Despite the limitations of the examination that are mentioned in table 3.1, we feel confident about basing our treatment of unknown words upon it. We observe that the ratio of letters/syllable in a word has a clear relation to the word's length; words with four or five letters have a comparatively high ratio—above 3.6—while long words, such as those with fourteen letters, have a low ratio—below 2.8. In light of this rather surprising result, we will implement the following approach: letting S_i be the mean letters/syllable observed for words with i letters, when we observe an unknown i-letter word, we will

Num. of Letters	Mean Letters/Syllable	Std. Deviation Letters/Syllable
1	0.9778	0.1217
2	1.995	0.07313
3	2.948	0.2759
4	3.66	0.7551
5	3.633	1.263
6	3.231	1.016
7	3.213	0.7676
8	3.106	0.7286
9	3.051	0.6655
10	2.97	0.6234
11	2.902	0.5465
12	2.871	0.5078
13	2.841	0.4575
14	2.802	0.438
15	2.792	0.4529
16	2.797	0.4197
17	2.828	0.4273
18	2.946	0.5268
19	2.388	0.1917
20	2.5	0
Total	3.164	0.836

Table 3.1: The mean letters/syllable, and the standard deviation from the mean, as obtained from the MRC2 database.

Note that each entry in the database received the same weight; thus, polysemous words are counted more than once and no attempt has been made to take the frequency of a word's use into account.

treat it as having *round*($\frac{i}{S_i}$) letters, where *round* indicates that we will round the result to the nearest integer. While this will introduce some noise into our calculations, since the MRC2 database is very extensive we expect that this will be insignificant.

3.5 Statistics on Sentence Length

According to Holmes [7], Yule proposed sentence-length as an identifying auctorial attribute as early as 1938. Despite its venerable history, even those scholars who have used this information point out that its usefulness is fundamentally limited: first, sentence length is often controlled consciously by an author; the most informative statistics, they argue, are those of which the author—and the reader—are unaware. While this criticism is valid and relevant to our work, it should be remembered that an author's choice of sentence lengths—like his choice of content words (content-word choice being viewed almost universally as highly informative)—says a great deal about how the author thinks and writes. A person's style, after all, is not determined merely by their unconscious, but by how their mind works in its entirety [16]. A second criticism of the use of sentence-length statistics is that sentence lengths depend explicitly on punctuation, which is often, particularly for pre-twentieth century works, determined by the editors of the work. In our corpus, the authors themselves have complete control of punctuation, so that this criticism does not apply to our experiment.

Most modern authorities agree that sentence length statistics provide confirmatory information, but conclusions should not be drawn from them alone. Further, there seems to be a consensus that sentence-length distributions provide valuable information, while average sentence length is not useful. The size of our samples makes it plain that calculating sentence-length frequencies is wholly pointless—it is extremely unlikely that sentences of any one length will occur more than once in any sample. Even the average sentence length will be subject to great variance, since we cannot expect our samples to contain more than five sentences on the average. Since this statistic should not be difficult to calculate, we will include it, if for no other reason than to confirm that it proves not to be useful in our testing.

3.6 Frequency Distribution of Punctuation

As was just mentioned, workers in fields related to stylistics have been very reticent of allowing punctuation to play any part in their studies, since this is often out of the control of the original author of the text under consideration. In our corpus, punctuation is completely determined by the original writer, and particularly in view of the small size of our samples, we feel it would be highly detrimental to our work to deny ourselves possible information conveyed by punctuation. With a view to capturing some of this information, we will record the frequency distribution of several punctuation marks that can be expected to appear commonly in electronic text. Since there has been no work in this area, we have been obliged to choose the set of punctuation we will use on the basis merely of personal knowledge acquired by reading portions of our corpus and similar electronic sources. For our set of punctuation, we choose those characters listed in table 3.2.

Most of our choices of punctuation to examine require no comment. As we discussed in section 2.5, there is very little standardization in the e-mail domain as regards the use of quotation marks. This explains why we have included four different quotation marks in our list of punctuation. Even our list is not exhaustive: a single apostrophe is often used as a quotation mark, particularly to set off new or unusual terms. We believe that apostrophes are in general rather content symbols than punctuation characters—their most common use being to indicate possession and in contracting words—and, since we do not believe we can reliably distinguish occurrences of apostrophes as punctuation from other occurrences, we have chosen not to count them at all. For similar reasons we have

Punctuation Mark	Gloss
.	period; also used in ellipses; not when used to separate acronyms or as a radix point
?	question mark
!	exclamation mark
:	colon, also used in emoticons (e.g. :-)); not when used as a token for separating dates/times
;	semicolon
,	comma
—	dash, when used as a parenthetical mark
( )	parentheses; each occurrence counted; often used for emoticons
[ ]	brackets; each occurrence counted
{ }	curly braces; each occurrence counted
<>	angle brackets; less-than and greater-than symbols; each occurrence counted
"	non-directional quotation mark
“	opening double quotation mark (sometimes used as a non-directional quotation mark)
”	closing double quotation mark
‘	(rare) used for single quotation mark
/	separator; e.g. s/he, and/or; not when used as a token for separating dates/times

Table 3.2: List of punctuation marks we will compute frequencies of, complete with descriptions of their common uses.

The symbols for which the note “each occurrence counted” has been added usually come in pairs, but for our purposes each component of the pair will be counted separately.

omitted the percent sign "%", the dollar sign "$", the hyphen "-" and so on. We will also attempt to count the character "'" only in its role as a quotation mark, not as an accent. Accented words will be counted as separate types, and so their occurrences will be noted, for example, in our calculations of the richness of an author's vocabulary (see section 3.10 below). We should also note the care we took in dealing with dashes: a string of two or three consecutive "-" characters is always considered to be a dash, while a single "-" character is only considered to be a dash if isolated by whitespace characters.

Table 3.2 also makes reference to the fact that each component of a grouping-symbol is counted separately. While we do not deny that, logically at least, parentheses come in pairs and thus form a single unit of punctuation, since many parenthesis symbols have multiple meanings (parentheses often occurring in emoticons; < and > having mathematical meanings) and we have no obvious way of distinguishing one meaning from another, we felt that counting each component of a pair separately was justifiable. Also, it is well-known that some authors are much more fond of parenthetical phrases than others—this thesis, for example, being rife with them—so giving extra weight to this class of punctuation may help to highlight such tendencies. For similar reasons, we have chosen to count each component of emoticons, and interrobangs, separately.

3.7 Distribution of Parts of Speech

Various researchers have investigated frequencies of use of various part of speech (POS) categories in attempts both to distinguish different genres and to attribute certain works to particular authors. While some have contented themselves simply with looking at the frequency distribution, Holmes [7] cites a study by Antosch, who investigated verb-adjective ratios and found them to be characteristic of certain genres. Holmes [7] also cites work by Brainerd, who extensively studied article and pronoun frequencies and showed them to vary far more between genres than between authors.

Although no one appears to have applied these techniques to samples as small as ours, we will calculate relative POS frequencies for as many categories of words as our tagger(s) will permit. While information such as verb-adjective ratios might in some cases be highly valuable, since we are dealing with a single genre it would not appear that even this well-studied ratio would be worth including. Further, it should be possible to recover this information from the statistics we derive, should we develop increased interest in it. Indeed, the networks we train on our statistics may turn out to use such relationships, and such a result would be highly informative.

3.8 Function-Word Frequencies

The use of function-word frequencies—that is, the number of times some function word W appears in a text divided by N for that text—has a long history in stylistic research. Holmes [7] cites a study from 1962, as well as the famous work of Mosteller and Wallace in 1964, both of which involved function words. Researchers find function words attractive because their use is far less susceptible to conscious control than is the use of content words, so that their study may reveal more about the unconscious aspects of an author's style. Further, their extremely high frequency means that even in samples of a few thousand words, their frequencies are likely to be meaningful.

Little work appears to have been done using function-word frequencies on samples of the size we are considering, but we feel that they are certainly worth computing. To decide just which words to compute frequencies for, we first selected all words in the Kučera and Francis corpus [15] with frequencies above one thousand—all words which we can expect to observe with a frequency of at least one per thousand. There are 92 such words. We then used the MRC2 database to determine all parts of speech with which these words are identified. Unfortunately, the MRC2 database gives many archaic and obscure parts of speech—e.g. "one" is listed as a verb as well as a noun and a

K&F Frequency greater-than	Number of Function Words
$20,000$	6
$10,000$	7
$5,000$	18
$4,000$	22
$3,000$	29
$2,500$	34
$2,000$	40
$1,700$	50
$1,000$	66

Table 3.3: The number of words that we would consider as function words having frequencies higher than certain values in the one million-word K&F corpus.

pronoun, “an” as a preposition, a conjunction, and an adjective as well as “other”, the category including articles. Thus, we had to use considerable knowledge of common uses of these words to remove from the list all words commonly used either as verbs or nouns (excluding pronouns but including auxiliary verbs). We did this because we believe that these words, almost by definition, can serve as content words. Further, it is much more common in the literature to use only such categories as conjunctions, prepositions and articles as function words than such words as “would” or “man” that have a frequency in English above one in a thousand.

After this pruning process, 66 words remained. Table 3.3 gives the fraction of these words whose Kučera and Francis (K&F) frequencies are greater than certain values.

Considering the size of our samples, it does not seem reasonable even to compute frequencies of words whose K&F frequency is less than $2,000$—our samples contain, on average, far less than 500 words, so we expect each such word to appear zero times in each sample. We will compute frequencies for all forty words with K&F frequencies above

the	of	and	to	a	in	that	he
for	it	with	as	his	on	at	by
i	this	not	but	from	or	an	they
which	you	one	her	all	she	there	their
we	him	when	who	more	no	if	out

Table 3.4: The list of function words for which we will compute per sample frequencies.

2,000. While it is unlikely that the eleven words with K&F frequencies between 2,000 and 3,000 will appear an adequate number of times in our samples to be of use to us, since it is trivial to calculate frequencies for any word once we have developed software to calculate the frequency of a single word, we lose little in computing frequencies even for words we are unlikely to be able to use. We can ascertain during the next phase of our study which statistics are usable, even testing our networks on subsets of the set we will compute. The function words we will compute frequencies for are listed in table 3.4.

In order to investigate the extent to which our selection of function words is orthodox, we have taken three lists of function words (or "stop words" as they are called in the information retrieval community) from websites. All three lists are much more extensive than ours, but this is to be expected since, particularly in IR applications, having an extensive knowledge of what words should be ignored is very important. The list published at `http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words` was a proper superset of our list. We found that the list found at `http://www.sbl-site-.org/SWISH-E/AutoSwish/SBL_wordlist.html` was a superset of ours except that it did not contain the words "I", "not", "or", and "all". When we observe that this list was compiled for biblical applications, where Roman numerals are important, the absence of "I" is unremarkable. It is somewhat more surprising that the other words were considered to be content words, but this must indicate that, in the judgment of the compiler, these words are important in biblical research; it seems clear, though, that the decision was very domain-specific. More surprising was our discovery that the list compiled by Mitton, from his paper on spelling checking [22], does not contain “I”, “that”, or “more”. Mitton's interest in spelling correction may explain his decision to treat “I” as a non-function word; we cannot, however, determine why “more” and “that” are absent—particularly in view of the fact that words like “moreover” are included as function words. In spite of our slight disagreements with Mitton and the creators of the SBL project, we believe that we have demonstrated that our selection of function words does not differ significantly from standard practice.

As with some other stylistic statistics, it has been found that individual authors' characteristic function-word frequency distribution changes during their lifetimes. Lancashire [16] points out that this occurs in the works of Shakespeare, for example, and many other authors have noted similar trends. Again appealing to the speed with which e-mail articles tend to be written, as well as to the fact that our goal does not require us actually to ascribe text to authors, we have no need for concern about this potential limitation of function-word frequency distributions.

3.9 The Type/Token Ratio

The definition of the type/token ratio follows directly from its name. Usually denoted by R, it is defined as

$$R \stackrel{\text{def}}{=} \frac{V}{N}. \tag{3.5}$$

Interest in R focuses mainly on its purported capacity for communicating something about the extent of an author's vocabulary. If one imagines an author to have some stock of words, from which the author chooses some words more often than others, one may be able to get an estimate of the size of that stock by looking at the number of different words introduced over some sample of text. If an author has a large vocabulary, from which he/she can select many words, R might be expected to be large; otherwise, R will be small. This is a very tractable statistic—it has historically been particularly easy to calculate it for works of interest, since concordances for various works were constructed in ways that facilitated this calculation, and made it even feasible by hand.

There are obvious problems with this simplistic characterization of vocabulary richness. The most significant is its explicit dependence on N, and the fact that while N can effectively increase without bound, there being only a finite number of words in any language, V cannot do so. Thus,

$$\lim_{N\to\infty} R = 0,$$

independent of any other factor. This drawback is fatal for our corpus, since our samples are not only small—making R unstable—but vary comparatively widely: we have paragraphs with ten words and paragraphs with five hundred.

Fortunately, we believe we have found a novel technique to solve the problem of varying sample lengths. As will be discussed in section 3.11 below, great effort has gone into developing statistical models of authors' characteristic vocabulary distributions. Holmes [7] cites some interesting work by Muller which seeks to remedy the difficulty of comparing the vocabulary distributions of texts of different lengths, and we believe we can bring this solution to bear on the type/token ratio problem.

We first note that, if we reduce a text's length by some fraction

$$U \overset{\text{def}}{=} 1 - N_f/N \tag{3.6}$$

where the quantity N_f represents the length of the text after reduction, the probability that a type appearing once will disappear is U, provided the reduction is made randomly. The probability that a type appearing twice in the text will disappear after a random length-reduction is U^2; in general, the expected number of types that will disappear from the text after reduction, V_d, is

$$V_d = \sum_{i=1}^{\infty} V_i U^i \tag{3.7}$$

and thus the expected value of V, V_f, in the reduced text—that is, the expected number of types that remain—is given by

$$\begin{aligned} V_f &= V - V_d \qquad (3.8)\\ &= \sum_{i=1}^{\infty} V_i (1 - U^i). \qquad (3.9) \end{aligned}$$

Application of this formula requires a complete knowledge of the frequency distribution of lexemes. Since such knowledge will be useful in computing other statistics, such as the lexical entropy discussed in section 3.2, and since it is not in any event a difficult programming task given the resources at hand, we do not view this fact as a significant drawback.

It can be shown that the ratio V_f/V is a function only of U, N/V, and V_1/V. Holmes [7] notes a study by Ratkowsky and Hantrais, who computed tables listing V_f/V for various values of these three ratios; they found that for large V_1/V or low U, V_f/V is virtually independent of N/V. While the validity of this result does depend on the Waring-Herdan distribution (see section 3.11 below) providing a good fit for V_i, since we expect that V_1/V will be relatively high in our small samples, and that, in most cases, U should be low, they do increase our confidence in the method.

We propose to combine Muller's text-length reduction technique with the type/token ratio as follows: for some appropriate value of N_f, we will compute V_f on all samples where $N \geq N_f$. We will then compute and record the ratio V_f/N_f.

We must now turn to the question of the choice of N_f. As of yet, we have no statistics on the size of our samples. Therefore, we cannot give a meaningful value for N_f, except to say that the distribution of sample lengths will be such that N_f will be greater than N for some proportion of our samples. For these samples, we will use an average of the statistics computed for the samples for which we were able to compute statistics, since this seems to be the only well-motivated means of ensuring that we have statistics for each sample.

We should note that we could simply have picked N_f words at random from our samples for any N_f, then computed V_f on the resulting set. After all, it is the case that the text reduction technique was designed primarily for situations where researchers have concordances, and thus random-sampling would have been infeasible since new concordances would have had to have been created. This suggestion has two drawbacks, however: first, text reduction allows us to use all the information contained in a sample, rather than ignoring a substantial and possibly valuable proportion of the sample's contents. Secondly, since we cannot take a truly random sample of text, this method appears to be theoretically better-motivated than taking an inevitably biased part of text out of our samples.

Holmes [7] reports that Baker developed a statistic which he called "pace", defined as R^{-1}—to be thought of as the rate at which an author uses new words. Baker claims good success with this statistic—success independent both of text length and genre. Holmes properly expresses suspicion of this result, considering that Baker's work has no statistical foundation and it is based only on comparing Shakespeare and Marlowe—whose works are of similar length and genre. Although very easy to calculate, we believe that even if Baker is correct our sample sizes are simply too small for this statistic to be unbiased. Far more effective methods of estimating the richness of an author's vocabulary have been developed and are discussed below, so it would not be profitable for us to compute R^{-1}.

3.10 Measures of Vocabulary Richness

3.10.1 Simpson's Index

More sophisticated techniques than the basic type/token ratio have been tried in the attempt to measure the richness of an author's vocabulary. Holmes [7] cites work from 1949 by Simpson, who proposed that a good way to characterize the richness of the vocabulary used in a sample would be to measure the chance that two arbitrarily-chosen word-tokens would belong to the same type. Denoted D, this is equivalent to the ratio of the total number of identical pairs divided by the total number of possible pairs; that is,

$$D \overset{\text{def}}{=} \frac{\sum_{i=1}^{\infty} i(i-1)V_i}{N(N-1)}. \tag{3.10}$$

Since D measures the degree with which words are repeated, not just the number of words in a given sample, it is far more reliable to work with than R. From Holmes [7] we learn of work by Johnson who showed that

$$\mu(D) = \sum_{i=1}^{\infty} P_i^2 \tag{3.11}$$

where $\mu(D)$ here denotes the expected value of D and P_i has the same meaning as in equation 3.2. The above equation means that the expected value of D is the probability that any two items chosen at random from the entire population of vocabulary words will belong to the same type. Therefore, D is seen not only to measure richness but to provide an unbiased measure of the corresponding population value, independent of N. It is thus clear that D could provide us with important information and we will compute it.

It has been shown that D is far more sensitive to variations of higher-frequency words—nearly always function words—than lower-frequency content words. The almost negligible contribution of non-function words is very unfortunate from our perspective, since we would like to follow Lancashire [16], who recommends that content words contain much information about style. We will nonetheless compute D, bearing this caveat in mind when we examine whether D and our function-word frequencies appear to play different roles in our testing.

3.10.2 Yule's Characteristic

Holmes [7] describes work done by Yule in 1944, who developed an estimate of vocabulary richness based on the assumption that the occurrence of a given word is based on chance and can be regarded as a Poisson distribution—that is, the likelihood that some given word will be used is fixed over any amount of time. Under the Poisson assumption,

$$\Sigma P_i^2 = \sum_i i(i-1)V_i/N^2.$$

From this, Yule derived his "characteristic":

$$K \stackrel{\text{def}}{=} 10^4 \frac{\sum_i i^2 V_i - N}{N^2} \tag{3.12}$$

where 10^4 is used to scale the output. It can be shown that

$$10^{-4}K = D(1 - 1/N).$$

Although several modifications to K have been proposed in the literature, none appears to give substantial advantages. Some authors propose that K should be calculated using only some classes of words, but this too has not met with general approval in the literature, most scholars preferring to include all words when computing K. Holmes [7] cites work which showed that, under the Poisson assumption, K is constant with respect to N—a very desirable property from our standpoint. Nonetheless, many authorities are still skeptical of K, since the notion that word selection from an author's vocabulary can be modeled by a Poisson process is not well-accepted.

We must take care to ensure that the values of our statistics do not deviate dramatically from the range 0 to 1, since this could introduce bias into our networks; it is well-known that neural networks are not tolerant of input that varies widely in magnitude. We will therefore compute $10^{-4}K$, since the value of this statistic will not differ much from D and thus will not be outside our preferred range. We feel that we should compute a version of K because, although closely related to D, there is some evidence that it has produced useful information, and we should therefore at least evaluate its effectiveness.

3.11 Vocabulary Distribution Modeling

Even to one completely unfamiliar with statistical characterizations of style, to attempt to describe the word-usage patterns of any person with a single measure—however well-motivated and thorough—must appear astonishingly crude. Practitioners in the field agree, and so have expended much time in developing methods for modeling word-usage. Chief among these is the modeling of vocabulary distributions. This is based on the idea that for all authors, there exists a parametrized relationship between i and V_i, and that particular values of the parameters are associated with particular authors. This relationship is usually modeled as a distribution—that is, researchers have sought distributions which appear to correlate well with the values of V_i empirically observed in many works.

Many prospective distributions have been tried. The best results have been achieved by the Waring distribution, applied to this problem by Herdan in 1964 and now known in the stylistic literature as the Waring-Herdan distribution. The distribution has two parameters, x and a, such that

$$0 < a < x;$$

its formula is derived by multiplying $(x-a)^{-1}$ and a series expansion of $(x-a)^{-1}$ by

$x - a$, producing the expression [1]

$$\begin{aligned} 1 &= \frac{x-a}{x} + \frac{(x-a)a}{x(x+1)} + \frac{(x-a)a(a+1)}{x(x+1)(x+2)} + \dots && (3.13) \\ &+ \frac{(x-a)a(a+1)\dots(a+n-2)}{x(x+1)(x+2)\dots(x+n-1)} + \dots && (3.14) \\ &= \frac{x-a}{x} + \sum_{i=2}^{\infty} \frac{(x-a) \times \prod_{j=0}^{i-2}(a+j)}{\prod_{j=0}^{i-1}(x+j)}. && (3.15) \end{aligned}$$

We then assume that, in the limit of an infinite text size, the value of the i-th term of this series will approximate the proportion of types that appear i times in the text. That is,

$$\begin{aligned} P_1 &= \frac{x-a}{x} && (3.16) \\ P_i &= \frac{(x-a) \times \prod_{j=0}^{i-2}(a+j)}{\prod_{j=0}^{i-1}(x+j)} \quad 2 \leq i && (3.17) \end{aligned}$$

In order to make this model useful, x and a must characterize the length of text as well as the richness and extent of its vocabulary. Herdan proposed the following formulae:

$$\begin{aligned} a &\stackrel{\text{def}}{=} \left(\frac{V}{V - V_1} - \frac{V}{N} - 1\right)^{-1}, && (3.18) \\ x &\stackrel{\text{def}}{=} \frac{aV}{V - V_1}. && (3.19) \end{aligned}$$

Holmes [7] points out that Dolphin and Muller demonstrated that Herdan's original paper contained a calculation error, so that the formula for a should in fact be

$$a \stackrel{\text{def}}{=} (1 - \frac{V_1}{V})(\frac{N}{V} - 1)(\frac{NV_1}{V^2} - 1)^{-1}. \qquad (3.20)$$

From these formulae, the expected number of words employed i times in a text will be

[1]Holmes's paper gives the first term of this series as $\frac{x-a}{a}$. Since for most values of x and a this yields results > 1 we have assumed this to be a typographical error and so have used $\frac{x-a}{x}$ as our first term.

$$\mu(V_i) = VP_i \tag{3.21}$$

as P_i has been defined in equations 3.16 and 3.17. Thus, by examining a corpus of an author's work to compute values for a and x, we can generate values of $\mu(V_i)$ and compare these values with those actually observed in a text whose authorship is disputed.

Holmes [7] observes that Muller in 1969—even before the correction to Herdan's formula had been made—showed that this model is reasonable for text lengths

$$100 < N < 100,000.$$

This result is very favourable from our standpoint. Also, it has been observed that this distribution overestimates the contribution of low-frequency words, but even this is not necessarily a drawback of the technique for our purposes, since the contribution of high-frequency words will already be well-represented in our computations by the statistics we compute for function-word frequencies, and word-length statistics, for example. However, it has also been observed that, particularly for small samples of text, V grows proportionally with N. Thus, because our sample sizes vary significantly, it will not be appropriate for us to compare values of $\mu(V_i)$ for any samples. We can of course compare values of P_i for different values of i with every expectation of discovering valuable information—these are probabilities and are independent of text length. Recognizing that our samples are small, and thus we expect V_5, for instance, to be negligible, we will record only the first five terms of the Waring-Herdan distribution for each sample.

3.12 Special Type Frequencies

It has been estimated that ten percent of English vocabulary accounts for ninety percent of all English text. In spite of this fact, whenever the frequencies of types within a text have been analyzed, types appearing once (V_1 or hapax legomena) are always the most numerous group. Many researchers have postulated that valuable information about an author's style can be found by examining hapax legomena. Some research has been conducted into whether once-appearing words occurring at the beginning of a sentence could prove helpful. While the criticism that this depends on punctuation, often beyond an author's control, does not apply in our case, subsequent research has questioned the statistical soundness of this technique.

It has also been reasonably postulated that hapax legomena should increase, obtain a maximum, and then decrease as $N \to \infty$. However, at least two large studies have cast doubt upon this conjecture: Holmes [7] states that, for the two million-word Kierkegaard corpus, $V_1/V = 0.4359$; for a corpus with approximately 103 million words of American English, $V_1/V = 0.4472$. This surprising degree of constancy adds support to the contention that hapax legomena could play an important part in stylistic research. Holmes [7] cites Honoré, who, in 1979, defined a measure

$$\hat{R} \overset{\text{def}}{=} \frac{100 \log N}{1 - V_1/V}. \tag{3.22}$$

This measure is designed to test the propensity of an author to choose new in place of previously-used words; the higher the value of $\hat{R}$, the richer the author's vocabulary. Honoré found this measure to be stable for text of size above 1300 words in Latin text. Since this measure has met with considerable acceptance in the literature, we will compute a version of it. As with Yule's characteristic, we must remove the scaling factor 100 so that the values of the measure are closer in magnitude to other values we compute. Even after rescaling the measure, we use it with caution, since our text sample sizes are dramatically different from those Honoré employed, and we must expect that the rate of hapax legomena we observe will vary dramatically between samples and in general will be far higher than commonly seen in previous work. We will also compute the ratio of hapax legomena to the total number of types, but we will use it with even more caution than we use Honoré's measure.

Another measure that has attracted attention in the literature is the proportion of hapax dislegomena in a text. Holmes [7] reports that where $1000 < N < 400,000$ this ratio has been found to be stable; tests on real data have found that, for a particular author, the ratio is virtually constant. It has been hypothesized that this constancy derives from the fact that, after an author has written about 1000 words, approximately as many words will be used a third time (leaving V_2) as will be used a second time (leaving V_1 and entering V_2). Unfortunately, it has also been noted that, below 1000 words, the proportion of hapax dislegomena increases rapidly. While our samples are almost universally less than 1000 words in size, since we must calculate V_2 for other statistics, we lose nothing by recording the value of V_2/V for each sample.

3.13 Character-Level Statistics

As we have mentioned previously, broad agreement exists within the fields of authorship attribution and stylistics that statistics on the level of the lexeme are the most appropriate for work in these fields. But several statistics at the level of characters, in addition to those we have discussed in sections 3.2 and 3.6, have been tried: some authors, including Mealand [21], have calculated the proportion of words containing certain letters in particular positions within words. The reasoning behind this measure is that, particularly in inflected languages like English and Greek, valuable grammatical information can be captured by simply looking at the positioning of certain letters in words. For example, in English, if the last character of a word is "s", the chances that the word is a plural form are very high; if the last two letters are "ed", the word is more than likely a past participle. Hence, the information provided by such statistics may well be useful in situations where little other grammatical information is available. However, there is no obvious well-motivated means of determining just which letters to look for or in what positions in a word they should be sought. In our study, we have POS frequencies; since this information will be both more reliable and more plentiful than letter-ratios, we need not avail ourselves of them.

Another strategy that has been employed is to calculate the percentage frequencies of the letters of the alphabet in which the text was written. Sometimes percentages of numerals are also included. A priori, it is no less creditable that this approach should be successful than that determining frequencies of function words should prove productive; but, unlike the case of the latter, there has been little evidence advanced in the literature to support the former approach. Therefore we feel comfortable in refraining from using it.

3.14 Inter-Sample Approaches

So far in this study, we have focused exclusively on what might be termed "intra-sample" approaches to stylistic analysis. The common element found in all such methods is that they view style—or whatever aspect of an author's idiolect they purport to measure—as being intrinsic to the work of an author, as something for which a meaningful characterization can be rendered by examining a specific sample of work. "Inter-sample" approaches, on the other hand, view style as a comparative quality; though not meaningful (or at least not describable) in isolation, the direct concept involves the comparison of two or more samples.

If our goal is to assist a compiler of a collaboratively-written document, then inevitably most of the information we provide will have to be in the form of comparisons. Fortunately, there is nothing in the intra-sample perspective which disallows comparison—it simply becomes a secondary process; first one acquires the profiles of some samples, then finds similarities and differences amongst the set. The inter-sample approach is much more direct, and therefore arguably more immediately useful. In this section, we examine some inter-sample techniques and review their applicability to our problem. We conclude the section with a discussion of how an experiment to test these techniques might work, as well as an examination of the reasoning behind our decision to proceed with an intra-sample experiment first.

One field where inter-sample measurement is the norm rather than the exception is information retrieval (IR). One of the principal problems in this domain is to assess the degree of similarity between two or more documents, or between a query and some set of documents (or document segments). Many possible ways exist for doing this. Among these, tests for matching words are a major category. Such tests are often strengthened so as only to include verbs and nouns, and weakened so that nouns match even if they are similar (as measured using WordNet, for example) and verbs match if they are in the same Levin class (that is, are semantically-similar and possess similar "alternations"; see Levin's book [17] for details). Hatzivassiloglou et al [6], in developing ways to extract summaries from a set of very short articles, have refined these methods and defined some novel "composite" metrics—methods which report a match only if two "primitive" elements match according to certain constraints. For instance, a constraint might be that two similar verbs precede by less than some fixed number of words two similar nouns. The paper goes on to exhibit some interesting methods for normalizing the results with respect both to sample size and to the frequency of the features under consideration.

Stylistic studies and information retrieval efforts are quite disparate fields, of course. Whereas in IR, the content of a document is of paramount importance, in stylistics it is the way the words comprising a document are put together that is important, and content is if anything a distraction. Thus, IR techniques certainly are not directly applicable to this—or any other—study primarily concerned with style. Nonetheless, work focusing on inter-document similarities might do well to try and adapt some of these measures. Such adaptation might, for instance, consist of weakening the composite features of Hatzivassiloglou et al [6] so that they are independent of content, and capture some deeper structural characteristic of an author's writing habits—they might look for the placement of adverbs with respect to modal or main verbs, for example. That such work is strictly comparative seems certain; it is not at all obvious how such traits as these could be intelligently conveyed through the examination of a single, isolated document.

We have discussed in section 3.9 a method of normalization which has traditionally been applied to comparing the vocabulary distributions of works of different lengths. For studies in authorship attribution, the process involves computing a distribution (such as the Waring-Herdan model described in section 3.11) based on corpora of undisputed authorship. V_f is then determined for each author under consideration, in such a way that the corresponding N_f coincides with the length of the work whose authorship is in dispute. At this point, $\mu(V_i)$ for the various prospective authors is computed and compared with the values of V_i that are actually observed.

This approach could be very powerful for our problem, since vocabulary distributions can paint a detailed picture of the richness of an author's vocabulary—undoubtedly a very important part of a characterization of an author's style. Indeed, we went to considerable effort investigating ways we might modify our experiment to incorporate this feature—and this was the genesis for our idea to apply text-reduction to the type/token ratio. We eventually were forced to conclude that, both because of problems with scaling and because there is no obvious means of integrating an inter-sample measure with our intra-sample measures, it would not be possible for us to include this type of measure in our study.

Clearly, it would not be difficult to construct an algorithm for finding boundaries in authors' contributions based on inter-sample measures. Further investigation of the above two techniques and other inter-sample methods will no doubt be necessary in subsequent studies. But we feel strongly—and the clear preponderance of intra-sample measures in the stylistic literature appears to support our contention—that style is fundamentally an intrinsic property of a sample of text, and that to give a human meaningful feedback on precisely how two samples differ it will be necessary first to characterize the style of each sample independently. While our hypothetical advice will no doubt have to be given in comparative terms, we believe that the most general and best-motivated approach to the problem is to compare well-founded measures of single samples, rather than utilizing measures based on some direct comparison of the samples. Such comparative measures may certainly provide valuable auxiliary information, but we think it unlikely that they will be sufficiently general or straightforward to be decisive in themselves.

3.15 Statistics Database Organization

The primary purpose of this section is to describe the format we use to store all the statistics that we compute in this phase. Nonetheless, it also serves as a good summary of those statistics.

As in the case of the database of authors' contributions we compiled in the first phase of the experiment, we feel that it is important for our statistical database to be organized in a way that makes the information as accessible to a human as to a computer. The design must also be sufficiently flexible to allow us easily to extract subsets of statistics that we might develop an interest in during the next phase of the experiment. With these goals in mind, we set considerations such as space-efficiency aside and focus on a design that is both simple and modular.

Our database will consist of a set of files each of which will be derived from and have the same name as some file from our processed corpus. A file will comprise several records, each corresponding to a particular paragraph of the corpus file from which the entire database file was derived. Each field of a record will exist on a separate line, and the entries (always numbers) in a field will be separated by one space character. The records will be separated with blank lines. Table 3.5 describes the order in which the statistics we have decided to calculate will appear in any given record.

Table 3.5 represents a daunting quantity of statistics—133 different statistics will

Line Number	Description of Fields within Record
0	paragraph number, num. of lines
1	N, V
2	average word length, freq. of i-letter words $1 \leq i \leq 15$ (words with length > 15 counted as 15-letter words)
3	average syllables/word, freq. of i-syllable words $1 \leq i \leq 6$ (words with > 6 syllables counted as 6-syllable words)
4	average words/sentence
5	rel. freqs. of various parts of speech
6	rel. freqs. of function words (see table 3.4 for details)
7	rel. freqs. of punctuation (see table 3.2 for details)
8	lexical entropy, Juola's measure
9	normalized type/token ratio R Simpson's index (D), modified Yule's characteristic ($10^{-4} \times K$) modified Honore's measure ($\frac{\hat{R}}{100}$)
10	ratio of hapax legomena ($\frac{V_1}{V}$) ratio of hapax dislegomena ($\frac{V_2}{V}$)
11	first five terms of corrected Waring-Herdan distribution

Table 3.5: The contents of each record of our statistics database.

Entries in the fields of the records will occur from left to right as indicated in the table. Line numbers are relative to the lines of each record—i.e. each new record begins with line zero.

be calculated for each sample of text. However, the organization of the records is not coincidental: we have designed the database so that certain categories of statistics will be easy to ignore, at our discretion. Indeed, it is unlikely we will construct any network that uses all the statistics—as mentioned above, we should likely avoid training our networks with both average syllables/word and relative frequencies of syllables/word.

The amount of work required to implement this table is certainly ambitious. But the function-word frequencies are trivial to calculate, as we describe in the next chapter. D and K are related so that once one can be calculated the other follows trivially. With these economies of scale in mind, we are confident that the set of statistics we have chosen to compute is both comprehensive and feasible.

Chapter 4

Generating the Statistics

4.1 Introduction

However straightforward and well-motivated a stylistic statistic may be, it can certainly never be a completely trivial matter to extract that statistic from a corpus of raw text. We have expended so much effort during this study on computing our statistics that we have seen fit to devote this entire chapter of this thesis to discussing these issues.

This chapter commences with an examination of the general approach we have taken in implementing and testing all of our statistical routines. It continues by describing some of the general assumptions we made concerning aspects of the text such as what measurements should be case-sensitive and what kinds of strings should constitute words. We proceed by considering some problems we faced when extracting specific statistics: namely, average sentence lengths and part-of-speech frequencies. We conclude the chapter by describing two simple methods we used to ensure that we had properly applied all of our measurements.

4.2 General Approach

From the outset, it was clear to us that a radically different implementation design was required for this phase than for the first phase of the experiment. Instead of performing a number of changes to a given file, our problem now was to produce a fixed set of data from every paragraph comprising a given file. We further observed that each data set would be composed of a series of subsets, corresponding to correlated statistics such as our various categories of frequencies; each subset would be generated by a discrete unit of code. Several of these statistical categories also depend on the same data: for example, both hapax legomena and hapax dislegomena require a knowledge of type frequencies, obtainable only via a complete knowledge of the tokens in a passage. Similarly, all of our vocabulary richness measurements require some information on the type distribution. The desire to avoid computing information multiple times for each paragraph while keeping our design as simple as possible motivated our decision to apply all our calculations in sequence to each paragraph, then record the computed data set, rather than to apply each operation to all paragraphs and then assemble all the data sets from their components.

As in the data-normalization phase of the experiment, we believe that the achievement of the highest possible accuracy must be at least as high a priority in this phase as the specification of a simple and efficient design. In order to ensure that our statistical routines are accurate, it is imperative that we be able to evaluate them separately as well as in conjunction.

As a means of accomplishing all these tasks, we used Perl's module facilities. Modules in Perl are the rough equivalent of classes in true object-oriented languages: they permit the storage of a set of methods and data elements, with mechanisms for controlling access to identifiers. Our module contains not only all the routines necessary to calculate each statistic from a given paragraph, but the routines to coordinate the sequence of processing and build a data structure containing all the statistics for a given paragraph.

It also contains the method to write this data structure to a file. Finally, the module contains all of our static data such as our function-word list.

The module paradigm provides an excellent means of ensuring accuracy because it allows us to construct a dedicated test platform for each routine which, after being thoroughly tested, can be used without change during the actual data-gathering run. For virtually all the functions in our module, we wrote small programs that took a paragraph on standard input, applied the routine of interest (with all necessary preprocessing) to that paragraph, and sent the output to standard output. This flexible structure allowed us to subject our code either to synthetic tests or real data with the same ease.

We have stored our data according to the description described in section 3.15. The data structure we used to store the statistics for a paragraph before committing them to a file is fairly straightforward, motivated by a desire to make the data-writing procedure as error-proof as possible. The data structure consists of an array of array references, each referring to an array containing the contents of a single line of output. For instance, the first reference in our data structure refers to an array containing the paragraph number and the number of lines contained in the paragraph—both, in this case, generated by the procedure responsible for coordinating the data processing.

The foregoing demonstrates that our desire to promote accuracy combined with simplicity and efficiency could be expected to be achieved by our design. As we make clear in section 4.4 below, it turned out that those goals were not all reachable for every statistical category. Unfortunately, we discovered this long after it would have been practical to alter our implementation enough to meet the actual situation fully.

4.2.1 Interesting Implementation Details

Before we justify several general assumptions, we turn to highlighting some of Perl's features that facilitated our implementation. One of the benefits of our use of Perl modules is their ability to perform certain actions when they are loaded—much as a constructor would do in a true OO language. By utilizing this feature we were efficiently able to load the MRC2 dictionary (over ten megabytes in size) into memory for our syllable-frequency calculations: instead of loading it for each paragraph, or each file, we were able to load it once per execution of our master data-generation script.

Perl's hash data type, or "associative array" as its manual terms it, is an invaluable tool in any work where arbitrary strings need to be associated with numerical values. We used hashes to store our function-word, punctuation and part-of-speech frequencies, as well as to keep track of the number of times each type had occurred in the paragraph under consideration. In a language without these facilities, it would have been necessary either to implement a hashing scheme of our own, or to use an array of data structures with both string and value fields, then to mount costly searching and sorting operations on this array. By using these hashes as parameters to procedures such as those responsible for vocabulary richness, lexical entropy, hapax, and Waring-Herdan statistics, we were able to keep our implementation both clean and modular. We have taken care to avoid using the facilities Perl provides for iterating through a hash when it comes to outputting our data; because of the way hashes are implemented, the order of this iteration is not defined. To achieve a consistent ordering of the frequencies we store in hashes, we created an array whose elements contain the keys of the hash in the order in which we wish its values to be output. Then we simply iterate through the array, using its values as the keys to obtain the desired hash value.

4.3 General Assumptions Behind the Statistical Computations

The implementation of most of the statistical routines was very straightforward; the next section describes those routines for which implementation was not simple. In this section we focus not so much on implementation-level details as on the assumptions we made about how the data should be treated for particular statistical categories.

4.3.1 Tokenizing the Text

Since almost all of our statistics are computed on the level of the lexeme, arguably the most important routine in our statistics module is that responsible for tokenizing a given paragraph. Not only must this routine be able to pick out all identifiable words from a given text, ignoring non-words like isolated ellipses or dashes, but it must be able to strip off punctuation from the words. This latter ability is crucial; for the purposes of our many type frequency statistics, "computer", "computer," and "(computer)" should all belong to the same type.

These considerations made it apparent that it would be necessary for us to write our own routine for tokenizing our corpus, there being no other software available that would meet our requirements. Our first step in tokenizing our text was to treat our paragraphs as single lines. This normalization simply makes implementing our algorithm simpler; since in this context newlines have no more interest than any other whitespace character, there are no disadvantages in replacing with them with spaces. We dispense with tab characters and multiple consecutive spaces in like manner; the presence of two or more consecutive whitespace characters is of no more significance to us than a single one. After this preprocessing, our paragraph is reduced to a single line of tokens, which are each members of any of several word-classes, and are separated by characters which do not fit into the word-classes given the context. Since all of our word-classes insist that a word commence with an alphanumeric character, to find all the words in the line we need only iterate the following two steps until the line is empty: first, we determine the longest series of characters starting at the beginning of the line which matches the specification of any word class. We then remove from the line those characters and all that come between them and the next alphanumeric character, or the end of the line. By assigning the substrings we determine to represent words to elements of an array, we have a straightforward means for identifying and extracting the sequence of words that makes up the paragraph under consideration.

We have identified eight word-classes. The most obvious of these is any string composed solely of alphanumeric characters. Thus, strings like “3D” or “A1” are considered to be single words. Any alphanumeric string strictly containing one or more single dash or underscore is also taken as a word; “3-D” and “run-of-the-mill” are therefore one word, but “B-” is truncated to “B”. While this treatment would be erroneous in the context of a report card, in our corpus examples of this usage are quite uncommon. Also, our corpus is not tightly proofread; it must be anticipated that authors will occasionally use a single hyphen after a word, instead of two or three, to mean a long dash; such punctuation should of course either not be followed by a space, or preceded and followed by one, but such small inconsistencies are often observed in the corpus. This lack of proofreading is beneficial to us in one respect, however: soft hyphens—hyphens inserted between two syllables in a word by typesetters to allow the word to span two lines and improve the aesthetic appearance of the text—are almost unknown in our corpus, and can therefore be safely ignored.

Alphanumeric strings with apostrophes in their last, second-last or third-last positions are also considered single tokens. They form our third word-class. Short forms like “’em” are not treated correctly, since the apostrophe is stripped; because such words are not common, but the use of this character at the beginning of words to signify a single quote is quite common, we felt this compromise was unavoidable. Unfortunately, this also means that when the apostrophe is used as a quotation mark at the ends of words, it will be included with the rest of that word; while this is not correct, the use of apostrophes in this position to indicate collective possession is very common indeed.

Acronyms are also considered whole words, and we permit both lower- and uppercase letters to comprise an acronym. Thus “a.m.” is considered as a single word. Since URL’s are very common in our corpus, they form our fifth word-class and we attempt to treat them as single words wherever possible; we include all characters between "http" or "ftp" and the next greater-than or whitespace character as part of the single token. It can be argued that this definition is too restrictive, since many URL's are not given with the protocol identifier; it can also be argued that, in view of their prevalence in our corpus, we should have specially provided for e-mail addresses as well. However, we were concerned that either broadening of the definition might have made it harder to treat strings such as "his/her" or "and/or" as consisting of two separate words—as they should be considered, in our view.

Numeric strings for both times (e.g. "12:45") and dates (e.g. "11/06/99") are considered single words, and are our sixth and seventh word-classes respectively. Finally, numeric strings containing one or more periods are also considered single words; not only are such often used for phone numbers, but the period is used in place of the slash as a date-separator by some authors. Periods which occur at the ends of these strings are not included: "3.14." is treated as "3.14".

As a result of the modularity of our definition of what strings constitute words, we were able to simplify our testing strategy by sequentially implementing the various word-classes, and then ensuring that they captured word-forms of their type yet did not interfere with each other. Even though we are confident that this simple strategy finds most problems, for such an important procedure it would have been beneficial had we been able to find a facility to give confirmation of the effectiveness of our procedure. Our first step in this direction was to examine the performance of the standard Unix utility `wc` (word count). This program outputs the number of characters, words, and lines in a given input stream. It has been part of standard Unix distributions for many years, and we had hoped to use its output to confirm that we had identified the correct number of words in a given paragraph. Unfortunately, the definition of "word" adopted by this utility is very simplistic: a word is viewed simply as a sequence of characters bounded by whitespace. Thus, the sequence `community -- there has been` is considered to have five words, while `community--there has been` only contains three. Since we believe that the true result for the above example should be four, it was clear that `wc` is too inaccurate to be of any use to us whatsoever.

Thus, we were compelled to rely on artificially-produced testing data, as well as a very small sample of real data. We conducted a small test of this routine in conjunction with our evaluation of our routine for calculating the average number of words per sentence in a paragraph. This test is summarized in table 4.1 located in subsection 4.4.1. We have found no significant errors in this routine, but it must be admitted that it has not been validated as extensively as might be desired.

4.3.2 Case Selection

A priori, it is far from obvious for which statistics the case of the words in our corpus should be deemed significant. We have made no assumptions in the statistical routines themselves; in none of them are any changes made to the input paragraph. Instead, we have chosen to express case-related assumptions in the statistical coordination routine, believing that such localization would make changing the assumptions in light of better reasoning a much less involved undertaking.

Of all the decisions we made along these lines, the decision to treat types case-insensitively clearly has the most wide-reaching ramifications for our work. All of our vocabulary richness and vocabulary distribution statistics, as well as hapax legomena and hapax dislegomena, depend directly on how V_i and V are related for various values of i. For instance, treating types case-sensitively would necessarily increase V_1 in proportion to V and decrease V_i for larger values of i.

While there seems little reason to prefer the case-sensitive approach—other than that it may accord better with a literal, or at least orthographic, interpretation of the definition of types—because of these extensive ramifications we shall provide detailed justification for our decision. We believe that vocabulary and case are orthogonal concepts. That some words, such as proper nouns, when written in English require capitalization plays no role in how an author might think about—or even think of—those words. It is also not relevant to when and whether an author might think to use words that sometimes require capitalization, when part of a cited title or when placed at the beginning of a sentence—indeed, there appears to be no category of words that may not appear at the beginning of any grammatical English sentence. Therefore, we believe that when attempting to evaluate the richness of an author's vocabulary—or any other characteristic attributed to vocabulary—case should play no role, even though words with letters of different cases are not, strictly speaking, the same.

For other applications than describing an author's vocabulary, case certainly is relevant. To deprive our part-of-speech taggers of the valuable information provided by case would doubtless have had dramatically negative implications for their accuracy. Likewise, Juola's statistic clearly considers case to be significant. Since it represents an attempt to characterize the structure inherent in a text, it could be argued that lexical entropy should also be measured in a case-insensitive context. However, since much of the structure of a text must exist in its division into sentences, we felt it better to measure lexical entropy with the case of the text unmodified. Naturally, function-word frequencies have been calculated case-insensitively.

4.3.3 Possessives and Clitics

Strings containing apostrophes posed some interesting questions. On the one hand, the strings "he's" and "isn't" contain words that we might like to count in our function-word frequencies. On the other, an author's choice to use "isn't" instead of the semantic equivalent "is not" definitely indicates something about the register of a passage, and possibly about the style of the author as well. Yet we wish to treat our data consistently with regard to all of our statistics.

In the end, we decided to leave tokens containing apostrophes unchanged for all our computations except those involving part-of-speech frequencies; see subsection 4.4.2 for a discussion of these issues. Clearly, for the purposes of our vocabulary richness measurements this is the correct treatment. For words denoting possession, the postulate that authors think of the word and its possessive function separately seems insupportable; the word “John’s” is a single entity, describing a relation of some thing to the concept “John”. This implies that “John’s” and “John” should be counted as distinct types. It seems reasonable to conjecture that people even think about words containing clitics like the modal verb negations as single units rather than as containing two distinct concepts. To use the language of Lancashire [16], we believe both these word-classes would act as one word in the author’s “cognitive black box”. By the same logic, it seems perverse to expand the clitic “n’t” to the word “not” and count it in our function-word frequencies. Thus, we are confident that our general treatment of words containing apostrophes is correct in the context of our experiment.

4.3.4 Juola’s Statistic

Before concluding our discussion of assumptions, we should describe our handling of the implementation of Juola’s statistic. For the purposes of this statistic, we have decided to treat newlines as plain space characters. There is a dual motivation behind this decision: from a practical perspective, it is much easier to deal with a single line of text than several lines, particularly when the “window” of Juola’s statistic would have to stretch between consecutive lines. From a theoretical viewpoint, when we recall that most text is generated with word processors and text editors with word-wrapping, we realize that most newline characters are inserted automatically, not by authors themselves; that is, most newlines come into being when an author presses the spacebar. Thus, we think that treating newlines as spaces in a character-level context such as the computation of Juola’s statistic is not only the simplest approach from an implementation perspective but is also the correct approach theoretically.

4.4 Problematic Statistics

Having discussed the general assumptions we made, and emphasized the ease of most of our implementation effort, we need to turn to two statistics that proved quite difficult to calculate, albeit for totally different reasons. In this section we discuss the obstacles we faced while attempting to compute average sentence lengths and part-of-speech frequencies.

4.4.1 Calculating Sentence Lengths

The problem of determining sentence boundaries is much more challenging than that of appropriately tokenizing a text. Indeed, the most accurate approach according to Manning and Schütze [18], a maximum entropy model developed by Ratnaparkhi, achieves results that are significantly less than perfect. We had not realized the full extent of this difficulty, and this underestimation may lie at the root of some of our difficulty in solving the problem.

In light of the perceived success of our approach to tokenizing our text, we tried to solve this problem in the same manner. We view a paragraph as consisting of a sequence of sentences, where each sentence can be recognized by a regular expression. Thus, to compute the average number of words in the sentences comprising a paragraph, it suffices to iterate through the following steps until no more sentences can be found: first, the minimum-length substring beginning at the start of the paragraph that matches our regular expression-based sentence definition is determined. The characters of which this substring is made up are then removed from the paragraph, along with certain "garbage characters" that may follow. The current sentence is then sent to the word-identifying procedure, which is used to obtain a count of the number of words in the sentence. If this number is determined to be nonzero, we increment variables counting the number of sentences and the number of words in the paragraph; otherwise the "sentence" is ignored.

Probably the most obvious weakness of this algorithm is that it is not obvious that English sentences should in general be recognizable by a regular expression, even if all we ask is that the regular expression be able to recover sentence boundaries. Our conclusion after this exercise is that it is indeed not always possible, though with sufficient effort it can be done to a reasonable approximation. Central to the difficulty of this problem is to distinguish when a marker that may indicate the presence of a sentence boundary is in fact being used for some other purpose. In English, the three primary sentence boundary (or “full-stop”) punctuation marks are the period, exclamation mark, and question mark. While the latter two are reasonably well-behaved, only infrequently appearing within sentences, the period is incredibly polysemous: it is used as a decimal point, to separate letters in acronyms, to separate date and even time fields, in short forms such as those for the names of months, and in titles such as “Dr.”—to mention only a few of its uses. Thus, the determination of sentence boundaries is fundamentally context-sensitive.

To disambiguate these situations, we have adopted the following approach. A sentence consists of a nonempty sequence of “units”, and ends when a “full-stop” character is found which does not fit into any class of unit. A unit is composed of a possibly empty sequence of non-word characters (any non-alphanumeric character except dashes and underscores) followed by a word. We only search for a subset of our word-classes here—only those classes which may contain periods, plus the general class of words that do not contain periods. We make no provision for apostrophes at all; the apostrophe is simply a non-word character that divides two words; it is not the responsibility of this procedure to precisely count words, only to segment paragraphs into sentences. We have also provided a generous set of non-word characters that may lie between a full-stop character and the last word character (e.g. quotation marks, right parenthesis, etc.) This set does not include space characters. This decision was based on our observation that, for our domain at least, when ellipses are isolated they tend to represent gaps within a sentence, while when they immediately follow a word they tend to be used as full stops. While this heuristic is not well-motivated, there is no question that ellipses are often used in both ways and it provided at least some means of distinguishing their function.

It should be noted that we are by no means the first to employ methods of this sort for this task: Manning and Schütze [18] describe a context-free method only slightly more complex than ours that has been employed to quite good success. Though we developed our method initially independently, we might have incorporated this version entirely if we had felt confident that its assumption that case and spacing are good sentence-boundary cues was valid for our domain.

If it were not for the fact that paragraphs must come to an end, and yet cannot be expected to end in a sentence terminator, even the primitive method described above would have had no significant problems. For example, if a paragraph were to end in a right parenthesis, the above procedure would enter an infinite loop: it would recognize the sentence boundary before the parenthesis, then be unable to match on the parenthesis since no alphabetic characters are present; the regular expression would thus leave the paragraph unchanged—and nonempty—and the loop condition, demanding an empty paragraph, would fail. Similarly, if no sentence termination character occurred between the last character in an identifiable word and the end of the paragraph, the match would fail and the loop would also execute infinitely. Allowing the end of the paragraph to serve as a full-stop indicator is therefore imperative. However, we must be very careful to ensure that no other sentence terminator lies between the beginning of the string and the end of the paragraph—else we will fail to count a legitimate sentence.

By using the "non-greedy" pattern matching facilities that are built into Perl and allowing the end of the paragraph to serve as a sentence terminator, we can effectively solve the latter problem. By "non-greedy" we mean that a pattern is matched to the minimum extent possible—in this case, until the first sentence terminator is found—instead of matching as much of the string as possible as is normal with regular expressions. The first problem is considerably more difficult to solve. We approached the problem with the belief that it was possible to define a reasonably small set of characters and character strings that may follow a sentence in English; examples are right parentheses, right brackets, quotation marks of all sorts, and for our domain emoticons of various sorts.

The data proved that this presumption was quite unjustified. After developing what we felt was a very generous set of allowed characters and character strings, and testing our routine with much artificial and some real data, we set it aside and completed our other work. When we ran our master data-generation script on all our files, we found that after processing a few files our program would enter an infinite loop, most often because it had encountered an example with some characters or character combinations between a sentence terminator and the end of the paragraph that we had not thought to include. Not wanting to regenerate our data because of the high cost of generation (see subsection 4.4.2 below), we simply fixed the specific problem and continued our generation from the point at which our procedure had been stymied. After quite a number of situations of this sort, we were forced to conclude that our approach is inherently flawed, and that we should have simply used Perl's built-in "non-word" character-class to eliminate all non-word characters between a sentence terminator and the next "word" character, (or the end of the paragraph), even though Perl's "non-word" character-class is rather artificial. But by this time, we were quite concerned that such a dramatic change would have an unknown and possibly significant effect on the consistency of our data, so we felt bound to continue the time-consuming method of simply fixing problems as they arose.

After having run through the entirety of our data set, we are confident that the average sentence-length data are at least highly consistent, even if their acquisition was quite costly in terms of time and effort. Table 4.1 shows the results of a test we performed on the completed average sentence-length routine (i.e. the routine that emerged after all data processing was complete). This table shows that the routine is quite accurate; indeed, the only errors we found that it made in this test had to do with counting numbers used in point lists as separate sentences, instead of counting them with the sentence they refer to. Even this error seems debatable. The table also gives us more confidence in our word counting capabilities, since on only two occasions in this twenty-paragraph test did our word-counting routine err, and on both cases it broke only one word up into two parts.

After completing the implementation of this procedure for computing average words per sentence, we commenced investigating the computation of part-of-speech frequencies (discussed in detail in the next subsection). As noted there, one of the taggers we used (that by Ratnaparkhi [24]) requires that its text be preprocessed by breaking it up into one sentence per line. The maximum entropy implementation of the sentence-boundary finder mentioned in Manning and Schütze [18] is included for this purpose. While Ratnaparkhi's sentence-breaking utility is considerably more efficient than his part-of-speech tagger, since our approach does not require the invocation of Java and all the overhead of executing Java bytecode, it is still considerably more efficient. Since it was not until much data had been generated that we realized our approach could not be made to succeed for all data, we felt that to switch to Ratnaparkhi's sentence-breaking program would introduce even more potential inconsistency into our data than simply discarding all "non-word" characters after sentence-terminators—the approach that would have removed our program from the danger of hitting an infinite loop. By the time circumstances forced us to regenerate a considerable portion of our data, our current procedure had been made sufficiently stable to withstand all of our data, so changing it was even then not a reasonable option.

4.4.2 Finding Part-of-Speech Frequencies

Determining part-of-speech frequencies for a text involves several steps. First, one or more part-of-speech taggers must be chosen. Then a method to send the text to the tagger(s) must be defined. Finally, if more than one tagger has been used, a means

Actual Values			Derived Values		
Num. of Words	Num. of Sentences	Average Words per Sentence	Num. of Words	Num. of Sentences	Average Words per Sentence
81	1	81	81	1	81
58	2	29	58	2	29
85	5	17	85	5	17
113	7	16.1429	113	7	16.1429
39	3	13	39	3	13
75	3	25	75	3	25
25	1	25	25	1	25
70	4	17.5	70	4	17.5
36	1	36	36	1	36
10	1	10	10	1	10
93	1	93	93	1	93
94	5	18.4	94	5	18.4
77	3	25.6667	77	3	25.6667
132	7	18.8571	132	7	18.8571
57	2	28.5	58	2	29
112	10	11.2	112	14	8
55	3	18.3333	55	3	18.3333
121	5	24.2	122	5	24.4
22	4	5.5	22	4	5.5
27	2	13.5	27	3	9

Table 4.1: Comparison of the actual number of words, sentences and average words/sentence with values derived from our average sentence-length routine.
Results are from two paragraphs chosen (manually) from each of ten randomly-selected files. The "correct" results were also generated manually; the derived results are from the output of the average sentence-length procedure, which outputs one number—the average words per sentence in the sample. Thus, considerable reconstruction had to be done to produce the derived data presented here.

of determining the optimal tag sequence must be implemented. In this subsection, we discuss our reasons for using multiple taggers and then examine in detail our solutions to each of these three tasks.

Choosing Taggers

Having examined the work presented by van Halteren et al [34], we are of the view that the accuracy of part-of-speech taggers on arbitrary text can be markedly improved by combining several taggers, particularly taggers based on different algorithms. Our text is not of the best quality, there being numerous typing errors and even grammatical errors throughout. Further, as we have been emphasizing throughout the study, accuracy is of paramount importance. Therefore, we felt that using multiple taggers would be worth the considerable extra effort that would be required, particularly in determining the optimal tag sequence.

By conducting a Web search for available software, we located and obtained source or executable code for five different taggers. The oldest tagger we found was the classic transformation-based tagger produced by Eric Brill [4]. This tagger was used by van Halteren et al, and is still considered to have a competitive accuracy. It is relatively simple to use, taking a text file as input and sending the processed file to standard output, appending to each word a slash and the appropriate tag, leaving the layout of the file otherwise unchanged. Unfortunately, the well-documented fact that this tagger expects to be presented with one sentence per line, and that punctuation be separated from words, was pointed out to us well after we had completed our data generation; hence, the data we produced with this tagger must be considered highly noisy. The version of Brill's tagger that we acquired had been trained using the Penn Treebank tag-set [25]. This tagger also has the advantage of taking relatively little time to load, and of producing output quite rapidly.

The chronologically next tagger we examined was the Decision Tree Tagger from the

University of Stuttgart [26]. This is a very robust tagger which is remarkably easy to use—one simply supplies the untagged text on its standard input and it sends the result to standard output. The time it takes to load is also by far the shortest of any of the taggers we examined; additionally, it is able to process a paragraph of text virtually instantaneously even on a comparatively slow machine. A considerable disadvantage of this tagger, which only manifested itself fully during the actual experiment, is that its output includes not only the words and the tags which correspond to them, but the word's morphological stem (or "unknown" if such cannot be determined) in an aesthetically pleasant columnar format, one line devoted to each word of the input. This adds very considerably to the quantity of output produced by the tagger. Our version of this tagger was also built using the Penn Treebank tag-set.

We have already alluded to Ratnaparkhi's maximum-entropy tagger [24]. This tagger assumes that the input text will be broken up so that each line contains precisely one sentence, so that in practice it needs to be used in conjunction with Ratnaparkhi's maximum-entropy sentence-boundary locator (supplied with the tagger). Both programs take their input on standard input and send output to standard output, sending copyright information etc. to standard error; thus it is simple to use them in a pipe to produce tagged text. As mentioned earlier, this system is monumentally slow: even on a fast machine (Pentium III, 500 MHz), it takes forty seconds of processor time to load; its text-processing speed is also very slow. Unlike Brill's tagger or the Stuttgart tagger, which are stand-alone programs, Ratnaparkhi's tagger is written in Java, and so must be run in a Java virtual machine; this contributes partly to its slowness. This poor performance seems mainly due to the size of the model the tagger employs: the program requires an eighty megabyte heap, which implies its model is very extensive indeed (and also that the tagger can only be run on a very powerful machine). Van Halteren also employed this tagger and its accuracy was impressive; the version we have was also trained with the Penn Treebank tag-set.

We also gave consideration to using the QTAG tagger [19] from the University of Birmingham. This tagger is only distributed in binary Java bytecode, and was expressly written for and trained with the Birmingham-Lancaster tag-set. While this tag-set is similar to the Penn Treebank tag-set, the differences are nontrivial. Since mapping one tag-set onto the other is a complex and error-prone task, and would have entailed the expenditure of considerable time, we decided to set this tagger aside without experimenting with it further.

The most modern tagger we experimented with was Scott Thede's hidden Markov Model tagger [31]. Thede has shown this tagger to achieve very high rates of accuracy; further, it can be trained using the Penn Treebank tag-set. Unfortunately, its design is focused on ease of testing rather than ease of tagging unknown text. This manifests itself most obviously in the fact that there is no way to train the tagger and subsequently use it: training and test data must be supplied simultaneously. We felt that no matter how we implemented our tagging procedures, training this tagger every time we wanted to use it would be computationally infeasible. We could have tried to modify the tagger's source code to give it the ability to store the results of training, but since we have three other taggers that are all relatively easy to use, and moreover which all employ different algorithms, we felt that the costs of this effort would far outweigh the prospective benefits.

Tagging the Text

Since all of our other statistics can easily be calculated on a per-paragraph basis, we strongly preferred to calculate part-of-speech frequencies in like manner. Because of the inclusion of Ratnaparkhi's tagger in our system, we realized that this decision would result in an incredible sacrifice of speed: of the total time taken to process any single paragraph, more than 97% would be spent simply loading, initializing and getting output from this single piece of software and its sentence-separating preprocessor. Since each paragraph takes approximately one minute to process on a fast machine (Pentium III with 500 MHz processor), with sufficient memory to cache all the programs and data, we realized that, even were we able to accomplish our data extraction in one pass, it would still require an entire week of processor-time.

Several problems would have confronted us had we chosen a more efficient method of tagging our text. The foremost is the problem of buffering in Unix interprocess communication. For purposes of efficiency, when a set of data is sent between processes, it is stored in a four kilobyte block of memory. Unless the process doing the writing specifies that the buffer should be flushed beforehand, usually after a line has been written, the data will only be made available to the process expecting them when the end-of-file is reached on the data-source or the buffer is full. If the buffer becomes full, the process producing it is suspended pending the buffer being emptied by the process to which it is being sent. For instance, if we had decided to tag our text file by file, we could not simply pipe the entire file to a tagger and then read its output, because all of our taggers (and Ratnaparkhi's sentence-processing utility) produce data as soon as they are ready. Had we tried to implement the above procedure, the tagger would be suspended as soon as it had filled its four Kb output buffer; this would cause our program to be suspended before it finished writing the entire file (all our files have size greater-than four Kb), thus resulting in "process deadlock". Unfortunately, while the taggers make data available as soon as they are ready they do not flush their output buffers; this means we could not simply send the file in a series of small chunks to the tagger and read the output of each: in this case, our program would wait for tagger output which would never come because the tagger's output has yet to fill its output buffer—another form of deadlock. Even sending the end-of-file character after each chunk does not necessarily avoid this problem; the only way to force a flush of the tagger's output buffer is to close the tagger's input pipe—terminating the tagger process.

Sending a file's contents as a tagger's standard input, but writing the tagger's output to a disk file might be one way of avoiding these problems. This process is inherently inefficient, although of course not as inefficient as loading Ratnaparkhi's code for each paragraph. It also forces us to make our process sleep until the file is completely written—a nontrivial programming detail. Finally, it offers no way to solve the second problem: given an entire tagged file, how does one unintrusively re-create the original paragraph boundaries? The most obvious approach would be to introduce a sentinel string into our input files, placing it between each paragraph. Such a string would have to be such that the tagger would not simply discard it in the output—i.e. it would have to be a string to which the tagger could attempt to apply a tag. This raises the highly complex question of what effect such "garbage strings" would have on the accuracy of the tagger: since they would occur very frequently, and be treated as normal text by the tagger, they would necessarily create an unknown and possibly significant level of noise in the data.

Particularly given the second difficulty, we felt that paying the onerous performance cost of executing all three taggers on each paragraph was the simplest solution—and the one giving the most accurate data. Even so, the problems noted above with interprocess communication dogged us throughout the data generation: especially in the case of the Stuttgart tagger, with its enriched output, a number of paragraphs caused the output buffer to fill before the input had been completely written—and thus deadlock ensued. The idea of using temporary disk files not having occurred to us at this stage (though we were compelled to use them in the case of Brill's tagger, which requires a file argument), we simply imposed arbitrary upper limits on the number of words contained in our paragraphs, truncating lines until the number of words in a given paragraph was under the limit. Fortunately, the percentage of long paragraphs is comparatively small: even for the Stuttgart taggers, for which we had to use the most drastic limit (between 230 and 240 words), we thus truncated less than 0.4% of the paragraphs in the corpus. Only seven paragraphs had to be pruned for all taggers. This certainly introduces noise into the data; since only a small number of paragraphs are involved we are confident that this very suboptimal approach will not skew our results drastically.

Determining the Optimal Tag Sequence

Unlike van Halteren et al, who were able to test the performance of their taggers against correct data, and were able to train "meta-learning" procedures in an attempt to find optimal weighting strategies to settle disagreements among taggers, we have no such resources. Thus, our only option is to treat all three of our taggers equally, and, when there is disagreement among the taggers, to use the tag chosen by the majority when there is a majority, choosing randomly among the three proposed tags when there is not.

This is indeed the strategy we implemented, but we were obliged to take into account some additional possibilities. There are situations when none of our tagger provides any meaningful tag for a given token. Of course, when at least one tagger provides a meaningful tag for the token—that is, a tag from the Penn Treebank tag-set—we pick the single valid tag (or choose randomly between the two proposed alternatives if only one tagger fails to return a valid tag). When no valid tag is returned, and we have no information at all, we have elected to treat the word as a singular common noun. This stems mainly from our observation that the ending of sentences by URL's is the phenomenon that most commonly brings this condition about, and in this situation the URL is certainly a singular noun (and, debatably, a common noun).

The problem of synchronization was a constant annoyance for us. This problem occurs when different taggers tokenize a segment of text differently—that is, when they have different ideas of what text units should be considered as words. Since we did not want to introduce noise into the data by changing them to accommodate the taggers, or to delve deeply into each tagger to understand the details of its tokenization algorithm, for the most part we adopted the following approach: if all the alphanumeric characters in the words returned by the taggers are the same, then we infer the tags apply to the same token and treat them normally. If we find that the alphanumeric characters differ, we assume that one tagger has tagged the text differently from the other two, which are still synchronized. Further, we assume that the unsynchronized tagger has merged n

tokens together, for some value of $n \geq 2$. We then look ahead to find the position where the merging ends—that is, where the alphanumeric characters comprising the strings returned by all three taggers agree. We then re-synchronize the unsynchronized tagger by inserting $n-1$ undefined tags between the tag for the merged string and the tag for the string that all taggers agree on.

This assumption turns out to be surprisingly realistic. In only one case did we encounter a situation which diverged irreparably; in that case, we had no option but to truncate the paragraph so that the text causing the unsynchronization to occur was no longer present. The decision to allow the unsynchronized tagger to vote on the first tag is largely arbitrary: it may well have been quite as reasonable to allow it to vote on the last tag in the sequence, or simply to throw the tag away entirely. We have some observational evidence that suggests this may be a slightly better approach, but this evidence is quite weak and circumstantial.

Regrettably, though this approach turned out to give acceptable results, at some point in our first round of processing the data we discovered a bug in our re-synchronization procedure. Since different taggers treat punctuation differently, we only wanted to compare alphanumeric characters. Our original version of the re-synchronization procedure failed to do this, resulting in many more re-synchronizations than necessary. In an examination of the data produced early in the first round of generation, we discovered that this error had been very significant, causing as many as 50% more tags to be introduced into a sequence than there were tokens. Thus, in spite of the enormous computational cost involved, we were forced to regenerate two-thirds of our data set in order to ensure data consistency; unfortunately we had kept only very imprecise records of when we had made this significant change, so erred on the side of caution by regenerating such a high proportion of our data set.

Despite our aversion to meddling in any way with our paragraphs, we were obliged to perform one more alteration for the sake of Brill's and Ratnaparkhi's taggers. The

Penn Treebank tag-set has tags for possession; i.e. words such as “John’s” have two tags applied to them: in this case, one denoting a singular proper noun, the next the presence of a possessive. The Stuttgart tagger takes this into account with raw text. Neither Brill’s nor Ratnaparkhi’s taggers do: they require that a space be inserted in front of all apostrophes, except in the case of the clitic “n’t”, where the space must occur in front of the “n” (so that the token may be tagged as an adverb). Abbreviations like “we’ve” are then also handled correctly, by having two tags applied to them. This procedure is easy to accomplish with two simple regular expressions, and we are quite confident that this has not introduced any noise into the data.

We were obliged to make several more unpleasant modifications to the text so that our resynchronization procedure could succeed. We had to remove all double dashes, used as point indicators, from the beginning of lines; one of our taggers ignored this sequence while the other two kept it, violating our assumption about how synchronization is broken. Double consecutive slashes needed to be removed for similar reasons. Isolated ellipses with more than three periods also needed to be truncated to only contain three periods. Brill’s tagger has a habit of entirely eliminating some classes of URL’s; to be safe we replace them in this case with the string “http”, so that Brill’s tagger is still presented with an unknown token. The Stuttgart tagger has an odd treatment of words surrounded by apostrophes; to avoid problems we simply eliminate the single quotation marks from its input. While all of these “kludges” are undesirable, none of them should distort any paragraph significantly, and we expect that in total their effect is negligible. Clearly, however, a more sophisticated synchronization procedure would have been desirable, if rather hard to implement.

4.5 Verifying the Output

To speed up completion of this very computationally-intensive task, we split our files into two groups and ran two similar machines simultaneously, one with each group. This, combined with constant crashes and restarts and compounded by the decision to reprocess the first two-thirds of our corpus that we had originally processed, made us feel it was necessary to check that no significant errors had been made in the mechanics of the data-generation phase. We did this in two ways.

First, we wished to see whether all files had been processed completely. Since, after a program crash (either due to a deadlock or an infinite loop), data would exist up to the last buffer write (we have not enabled output buffer flushing in our data-generation routine), there are only a very small number of places where a data file could end. Fortunately, none of these possibilities coincides with potential boundaries of records; that is, it is always the case that when an output buffer becomes full and is written to disk automatically, an incomplete data set will be written. Thus, to determine whether complete output files have been written, it is only necessary to look at the last two lines of all output files to see whether they match the last two lines of all data records. We found that this was always the case.

The other problem we feared was that, by mistake, during the second round of data generation we might have had our statistical procedures work on files from the first round instead of text files from phase I. To test this we simply observed that our data records always contain 137 separate tokens; we then merely had to look for the number 137 in the "number of tokens" field in the first record of each of our data files. Had we observed it, we would then have looked at subsequent records in the file to see whether they had this characteristic number in the same field. Fortunately, this did not happen.

We are confident that the data we generated in this difficult phase of the experiment are of very high quality. As the foregoing makes clear, we have sacrificed considerations both of efficiency and difficulty in order to maintain high quality data, and we feel we have in large measure succeeded. It is indisputable that approaches existed that were superior to the ones we employed at various stages of the implementation of our data-collection routines, and during data collection itself. This phase of the investigation has been illustrative not only of how unpredictable and varied real-world text can be, but also that one should at all times strive for good software engineering habits. Had we this task to do over, it is unlikely we would have generated all statistics at once, preferring the harder to implement but far more flexible approach of calculating each category of statistic for all files, then assembling all the results. But if we have not achieved efficiency or elegance, at least we have reached our coveted goal of a reasonably clean and consistent data set to use in the final, pivotal, phase of this experiment.

Chapter 5

Building Neural Nets to Locate Authorship Changes

5.1 Introduction

In section 1.4 we remarked on the growth of interest in the application of neural nets to authorship attribution problems, and more broadly throughout the community of researchers who use stylometric statistics. In this section, we briefly review some basic neural net concepts and terminology, then explore some of the reasons why neural nets have been hailed as promising in this community, with particular emphasis on qualities that make them useful for our research. We then briefly discuss the software we used in our experiments and the approach we took to training and testing. In each of the chapter's three final sections we describe a different neural net architecture and give the results we obtained while experimenting with that architecture.

5.1.1 Basic Neural Net Concepts

When neural nets were first proposed in the 1940's [32], they were viewed as a means of simulating the structures comprising the human brain. Just as the brain has neurons which send activations to one another across synapses of varying conducting capability, so neural nets contain "units" (nodes) which send activations to other units across "connections" (links) with varying "weights". In the brain, it was thought that learning occurred by a process of altering the conduction characteristics of synapses, so that when trained a brain would respond well to some given stimulus. The neural net analog was therefore obvious: to train a network, one simply needed to find a procedure for systematically changing the weights of connections between units so that activations caused by a certain stimulus would cause the appropriate response to be exhibited by the trained network.

Even though advances in our understanding of the brain have certainly discredited this simplistic model of human cognition, the terminology used to describe neural nets has remained relatively static. Modern neural nets are used for their statistical pattern recognition capabilities rather than for simulating the human brain. Almost all modern neural net architectures group units in sequences of two or more "layers". The first layer, the "input layer", is activated directly by data. The final layer is referred to as the "output layer" and produces the final output of the neural network for some data pattern. Between these two are zero or more "hidden layers", composed of units which are activated by lower layers and activate higher layers, and are present to account for nonlinear interactions within the data. It has been shown [3] that any neural net computation can be carried out by a neural net with a single layer of hidden units. Notwithstanding this fact, as we shall discuss below, more complex architectures are often preferred. Most network topologies require links only in a forward direction, but certain models allow units in the same layer to be interconnected—or even for the connection of units to themselves or to units in lower layers. Often, and always in our experiments, when connections exist from one layer to another, those layers will be "fully connected"; that is, connections will exist from each unit of the lower layer to each unit in the upper layer.

A unit's activation given a particular pattern on its input connections is determined by an "activation function". Activation functions first aggregate all the inputs to a particular unit (except in the case of input units where the feature of the input pattern the unit is connected to is used directly); in our case, the aggregation is always simply the weighted sum of the incoming connections to the unit. Formally, for the j-th unit in the upper layer, this aggregation is expressed as:

$$A_j = \sum_{i=0}^{n} w_{ij} \times O_i \tag{5.1}$$

where O_i is the output of the i-th unit in the lower layer that is connected to unit j (1 when $i = 0$), w_{ij} is the weight of the connection between the two units (or the "bias" of unit j when $i = 0$), and n is the number of units connected to unit j. Once this input value is determined, the activation function subjects it to some type of normalization; for all our networks, the sum is put through a logistic sigmoid function,

$$O_j = \frac{1}{1 + e^{-A_j}},$$

O_j is the final unit output, so that the output is forced to lie in the interval $(0, 1)$. We should emphasize that our use of the logistic sigmoid function in the output unit of each of our networks means that their output is always stochastic, never binary.

We only concern ourselves in this study with "supervised learning", or learning that involves the repeated presentation to the network of a set of input patterns and the corresponding correct outputs. Though paradigms exist that do not require the correct results to be known, or that only supply the network with an idea of how well it is doing overall, we believe that the supervised approach best suits our domain. Supervised learning algorithms require some "error function" for quantifying the amount by which the neural net's output differs from the expected output. All the learning algorithms we employ are "batch algorithms" (they change the network's weights only after evaluating the network's performance over the entire training set, as opposed to "online algorithms" which change weights upon seeing each component pattern). All our algorithms also use a "sum of squares" error function, so that

$$E = \sum_{k=1}^{M}(O_k - T_k)^2 \tag{5.2}$$

where T_k is the target output on the k-th pattern, O_k is the observed output, and there are M patterns in the training set. The various learning algorithms we employed differ in the way they apply this error function to changing the weights of the network. We describe them and our reasons for testing them at a high level in subsequent sections.

5.1.2 Why Neural Nets?

Tweedie et al [33] adduce several reasons for the increase in the use of neural nets among researchers who use stylistic statistics. In contrast to systems involving the definition of complex—and necessarily subjective—heuristics by the researcher or some expert in the area, neural nets learn directly from data. For our study this characteristic is critical; a priori, we have no clear notion even of how one might come to arrive at a set of rules for differentiating between authors' styles, let alone any specific set of rules in mind for testing. The problem is too complicated and multifaceted for such an approach to hold out much promise. Neural nets have a substantial advantage over more conventional classification techniques such as regression analysis in their ability to generalize. By modern standards, even our $750,000$-word corpus is not large; to make any claim of a generally-applicable result, we must avail ourselves of a technique which has proved successful at classifying data based on fundamental—and likely highly non-linear—interactions between input variables, not simply on details of the training set. Recognizing that our data are far from noiseless, replete as they are with unidentified quotes, mis-tagged words and un-removed signatures, our classificatory system must also be highly fault tolerant. Neural nets possess this property, having been known to train successfully even on quite noisy data.

Other even less traditional learning algorithms that display similar properties have been tried in this and related fields: Hatzivassiloglou et al [6] have used a rule-induction system, Holmes et al [8] genetic algorithms. However, an advantage of using a mature technique like neural networks is that much research and development will have gone into producing a large amount of high-quality software. A simple Internet search for neural net software yielded well over sixty separate hits. Even given our limitation to software that can run (or be compiled to run) on the Linux platform, the number scarcely diminished. After examining the high-level documentation for several packages, and reading various FAQ's such as that from the Pacific Northwest National Laboratory [2], we concluded that the best piece of neural net software available is the SNNS (Stuttgart Neural Network Simulator) system from the University of Stuttgart and the University of Tübingen [29]. This powerful, versatile and singularly reliable system alone lends considerable justification for our choice of neural nets for our classificatory mechanism.

5.1.3 General Course of the Experiments

Not only does SNNS support well over a dozen different neural net architectures, a score of learning algorithms and several different activation and error functions, it also contains a powerful scripting language that permits the automatic construction, training and testing of all manner of neural nets. It is to the power of this software and of the machines we ran it on that we owe the number and comprehensiveness of the experiments we were able to carry out.

All of our experiments have at least five phases in common: the creation of training and test pattern sets, the construction and initialization of the network to be tested, and the training and testing of the network. We have automated all of these phases, and we discuss them and the general evaluation strategy we employ in the rest of this subsection.

Many approaches have been used to evaluate how well a neural net is performing on a given problem. The most methodologically-satisfying involves the division of data into training, validation and test sets. The network is trained on the training set, which usually contains 80% of the data. Periodically, its performance is evaluated on the validation set, which usually contains 10% of the data. Once the error observed on the validation set reaches some threshold, or begins to rise as the network begins to overfit to its training data, training is stopped and the final performance of the network is evaluated by testing it against the test set, which contains the final 10% of the data.

Since, by modern standards at least, our corpus is not vast, we did not feel we could sacrifice 10% of our data to a validation set. Though other methods, such as jackknife testing [32] exist for conserving training data while allowing for some measure of validation, we have chosen the simpler approach of simply using a training set and a test set. While this might well be fatal to a study hoping to design a network to predict a response for real data for which no correct answer is known, our work is more at the proof-of-concept phase so that it is probably sufficient simply to exhibit a network which we can demonstrate performs well on data it has not been trained upon.

Before performing any of our experiments, we divided our data into training and test sets. Since all our networks are trained and tested on data derived from the same subsets of our corpus, the results obtained from them will be more directly comparable. Recalling our decision in the last phase to keep the data derived from different files of our original corpus in separate files, we chose simply to select 10% of the files (22 out of 221) for inclusion in our test set, leaving the rest for training. We used the same pseudorandom approach as we did when testing our quotation-identifying procedure, and it happened that the files we selected were larger than average, so that we ended up having 1729 data patterns in our test set and only 13829 pattern in our training set; i.e. our test set in fact represents 1/9 of our data.

Our decision to maintain the division of our data into files is motivated by several factors. First, each file is, in the broadest sense, not a completely artificial document;

that is, the articles have their original structure (minus quotations) and are in their original order, so that often articles on similar topics will be found consecutively, making our networks' task more realistic. To randomly pull paragraphs from anywhere in the corpus to assemble a test set would have destroyed this structure and compromised this element of realism—not to mention making our job of using the database of contribution boundaries that we computed in the first phase of the experiment far more difficult. Though our training and test pattern sets will be assembled into single files for all of our networks, the fact that they will be composed of a known set of files will make the task of relating our networks' performance to the original data much easier. Though we have not had time to examine any of our networks in great detail, should we have the desire and be in a position to do so this would constitute a significant advantage.

In SNNS, a complete pattern simply comprises two lines, the first line containing the input, the second the corresponding (target) output. We view an input pattern as consisting of a subset of the data we compute for each of two consecutive paragraphs in our corpus; we extend "consecutive" so that, if one file's issue number immediately follows that of another (as reflected in their file names) in either our test or training set, then the first paragraph of the former consecutively follows the last paragraph of the latter. Our output pattern simply consists of a binary value reflecting the absence or presence of a contribution boundary—that is, whether the consecutive paragraphs have been written by the same author or by different authors, respectively—as determined by examining our database of contribution boundaries. We are primarily interested in subsets of the data that correspond to some or all of the lines comprising our records for each paragraph. Hence, the script we have written automatically to generate the pattern files takes line numbers (refer to section 3.15 for details) of lines to be included in the input patterns as input, along with details such as the location of the data files and of the contribution boundary database. There are cases, particularly involving statistics for which we have frequencies and a cumulative average, such as syllable frequencies or word lengths, for which we would like to be able to disassemble individual lines. In spite of this suboptimality, this approach's ease of implementation combined with the fact that we have a plethora of other tests to perform convinced us that a more general approach would not have been practical. We are extremely confident in the performance of this script, never having seen evidence of an error in it.

In addition to its graphical user interface, SNNS contains command-line tools for generating large neural nets. In spite of their documentation, which is sometimes quite inconsistent—especially so in the case of time-delay neural nets, where we were forced to rely upon trial and error—we used these tools to create all of our networks. In the next three sections we will discuss the precise topologies and architectures we used these tools to evaluate.

Initialization, training and testing of the networks are handled very well by SNNS's built-in scripting language. Except in a few instances which we note below, all of our networks were initialized with weights randomly selected between -1 and 1. This is commonly done in neural net research when there is no prior information available as to what values weights should have; its purpose is to ensure that different parts of the network are affected differently by the data, so that learning is encouraged. The interval we have chosen is often selected because it allows for both excitatory and inhibitory weights—that is, it ensures that not all activations in lower layers will bias units in upper layers towards activation, so that the network can learn either to ignore irrelevant parts of the data or that certain parts of the data inversely correlate with the correct network output. On several occasions, we have used several different random initializations to train networks with the same topology. This is often done because, if the learning process is viewed as trying to find the global minimum on an "error surface", then the network may become "stuck" in local minima if learning is started from a suboptimal starting position. Thus, several different starting positions are often tried in hopes that the global minimum of the surface is reachable from one of them.

Commands are provided in the scripting language both to train the network with whatever learning algorithm is in effect, and to test the network. Our strategy at first had been to train the network for several "cycles" (complete propagations of the input data through the network and computations of the sum of squared error), then test the network and end the script. In all cases, we have used the "mean squared error", the sum of squared error divided by the number of patterns, as an indication of the network's performance; the smaller the MSE, the better the network. We eventually realized that our original approach to training was inefficient, since we had no information about whether our network was overfitting to the data. Therefore, for most of our later experiments we train the network for a fixed number of cycles, recording the MSE on each cycle, and then we test it, again recording the test MSE. We repeat this procedure some fixed number of times, or until we realize the network is overfitting to the data or is stuck, and we abort the script.

SNNS also permits the production of "result files", which contain the output produced by the neural net for all given input patterns in an pattern file. We use these result files to gain other indications of how well our network is performing. We have developed a script for calculating precision, recall, fallout, accuracy and error from these result files and their corresponding pattern files; we discuss some of the results we obtained using these scripts below.

5.2 Simple Multilayer Perceptrons

Whereas a network with no hidden layer is often referred to as a "perceptron", one with one or more hidden layers is a "multilayer perceptron". Largely because this approach was the most obvious, our first set of experiments involved a simple multilayer perceptron (containing one hidden layer) whose inputs were the entire data sets generated for two consecutive paragraphs. Despite our best efforts, which we recount below in hopes of learning something about the problem from the failure of such diverse approaches, we had very little success with this architecture.

One of the strongest criticisms that can be levelled against the neural net movement is that very often, designing a successful network is quite as much an art as a science. Even if one restricts oneself to a particular architecture, there are no hard-and-fast rules that dictate how the parameters—such as the number of hidden units—of any architecture can be optimally determined; even if one uses a pruning algorithm to eliminate useless units, one is certainly not guaranteed to find an optimal configuration. And when the architecture and topology have been fixed, there is an array of learning algorithms from which to choose, all making similar claims about speed and robustness, each coming complete with a set of parameters of its own that must be specified in advance. Therefore, to comprehensively evaluate this simple architecture, we have felt it necessary to experiment in all three dimensions—topology, choice of learning algorithm and learning algorithm parameters.

One of the oldest, and probably the most famous, learning algorithms used in neural nets is backpropagation. The algorithm is quite simple, conceptually: batch backpropagation simply involves propagating all the input patterns in the set through the network, computing the sum of squared errors for all patterns and adjusting the weights to attempt to minimize each of their individual contributions to the overall error. This minimization is effected by computing the derivative of the error function with respect to each individual weight in the network and using this information to determine each weight's contribution to the entire error. The algorithm's name derives from the fact that the computation of the weights' contributions to the overall error and their subsequent adjustment is made from the output layer back through the lower layers of the network. Interpreted spatially, the algorithm involves taking steps along the error surface in the direction of the steepest descent from any particular point.

Since backpropagation has been used for such a long time, researchers have had much time to make improvements to it. One reasonably simple improvement is to add a “momentum term” to the adjustments made to the weights. A particularly intuitive explanation for the effect this has is provided by analogizing the learning process with the progress made by a marble placed on a hilly surface. If the marble begins to roll down a long, gradual slope, it picks up speed. Conversely, if the marble begins to roll uphill, it quickly loses speed and its direction will change. If the marble encounters an area with sloping sides and a narrow bottom with a shallow downward slope, it will oscillate between the sides for a time but will gradually converge to the centre of the “valley”, and thus begin to make faster progress downhill.

Though there is a good mathematical underpinning behind this modification, we refer the reader to Bishop [3] or any other good text for the details. The intuitive description provided above, combined with the fact that this is a widely-accepted and well-known algorithm, should suffice to explain our preference of this algorithm as the first with which to experiment. Since our network had 266 input units—there are 133 statistics, in total, for each paragraph—we felt that 100 hidden units should be at least sufficient to produce some interesting results. Since the momentum parameter of the algorithm is the most critical, we made several experiments with different values of this parameter and of the learning rate; all of which, much to our chagrin, led within 25 cycles to a network which produced a constant output of between 0.212 and .270, depending on parameter values, regardless of the input! Though this produced a satisfying training MSE of 0.193 (see section 5.5 below for a comparison), and a test MSE of approximately 0.185, it was painfully obvious that a constant result was not desirable.

After pondering this problem at length, we realized this situation might be caused by weights with very large magnitudes within the network. Upon examination, the trained networks commonly exhibited weights with magnitude $> 10^5$, and sometimes as great as 10^9. When we recall that all our weights were initialized with magnitudes ≤ 1, it becomes clear that the momentum term introduced into the adjustment phase of the learning algorithm was producing vastly negative effects. Even after trying some relatively small values for momentum and learning rate (0.1 and 0.2 respectively), these symptoms persisted, so we abandoned the backpropagation with momentum approach for this particular network topology.

Another modification of backpropagation is designed precisely to counter situations in which training results in unreasonably large weights. This approach, "backpropagation with weight decay", introduces a "decay term" into the weight adjustment. This causes the magnitude of weights to be decreased automatically at each iteration of the algorithm. Adopting this learning algorithm and reducing the number of hidden units to 50 (it seemed clear that 100 hidden units was far too many), we trained a network for 1500 cycles with learning rate 0.3 and weight decay parameter 0.05. While the output of the resultant network was in fact influenced by the input, the training MSE rose to a staggering 0.22 and the test MSE to 0.217. Increasing both the learning rate and the weight decay factor slightly produced an even poorer network, with a training MSE close to 0.23 and a test MSE over 0.22. Returning the weight decay factor to 0.05, and decreasing the learning rate to 0.05 eventually produced a network with training MSE around 0.21 and a slightly higher test MSE. An examination of the weights of this network demonstrated that virtually all of them had decayed to virtually 0. This prompted us to train a network with weight decay parameter set to 0.005 with the learning rate still set to 0.05; this produced, after around 70 cycles, the slightly encouraging training MSE of 0.198. Reducing the learning rate to 0.01 and leaving all other parameters unchanged produced a training MSE of approximately 0.195 and a very encouraging test MSE of 0.181. Since this network was very far from overfitted, we increased the hidden layer's size back to 100 units, and received a training MSE of approximately 0.194 and a test MSE of about 0.182—results as easily explicable by the network having started from a different position as by the notion that 100 hidden units was better than 50. But since the values output by the network varied considerably—all the way from 0.2 to 0.33—we decided to further increase the size of the hidden layer to 150 units, only to find the network trained no better and the test MSE increased to 0.189. Admitting that moving further in this direction was pointless, we created a network with a slim 25-unit hidden layer, which trained to an MSE of 0.197 and a test MSE of 0.2012 after nearly 250 cycles.

These disheartening and ominous results can be explained in several ways. Most importantly, while even in the smallest network there are over $6,000$ separate weights, we have only $13,829$ patterns in our training set; clearly our networks are not nearly sufficiently constrained. While valid, this argument suffers from the fact that its logic would predict that our networks should overfit to the data, whereas in fact their training MSE's are almost always higher than their test MSE's. More convincingly, it can also be argued that this network architecture is simply insufficiently structured, that it completely obscures important aspects of the organization of the data, both in terms of their separation into two halves and into several internally cohesive sets (corresponding to the lines in which we store the data, as described in section 3.15) within and between those halves. With thousands of weights it is also true that the error surface will likely be extraordinarily complex, so that any particular starting position is likely to lead to an extremely suboptimal local minimum. Finally, as we noted above, the design of neural networks is not a clear-cut business; so these results may simply reflect the fact that we had yet to develop an intuition as to what topology and learning parameters best work in this domain.

5.3 Committees of Experts

5.3.1 Motivation

Bishop [3] discusses two related neural network architectures which he calls "committees of networks" and "mixtures of experts". By the first term he refers to groupings of identical (or closely-related) networks which have been trained on the same data from different starting weights, or with different learning algorithm parameters, and whose results are combined by some weighting system to produce an overall output. Such systems are used in cases where different network trainings yield networks which perform at varying levels, and is used in place of the usual practice of throwing away all but the best network in order to produce a system displaying better generalization characteristics. Mixtures of experts, according to Bishop, are collections of networks trained on different aspects (or subsets) or the data, whose outputs are used by a "gating network", which uses the entire data set to determine which expert is likely to be the most trustworthy given some input pattern. This network architecture is commonly used in situations where there is some structure to the data, but the interactions between various data components are not well-understood. Jacobs et al [9] describe a similar system where the gating network actively selects which expert it will consult for a particular data pattern, which also has the advantage of being useful in training, allowing changes to be localized to the selected expert and the gating network.

Our data sets certainly do contain structure; our various frequency statistics, for example, are obviously correlated. Thus, designing an expert network to operate on our data appeared to hold out substantial promise, particularly in view of the dismal performance of a simple multilayer perceptron. But neither of the above approaches was applicable: the first because our networks are already massive; to attempt to use several simultaneously would have been computationally prohibitive. The second approach is very interesting theoretically, but it would have been difficult to design the necessary network with SNNS—though SNNS supports the ability to combine several existing networks so that their outputs feed into another network, it has no facilities for allowing this upper network to have direct access to the data.

So we adopted a simple compromise. We trained ten separate "experts", one for each of the subsets of our data that we identified in section 3.15. Once trained, we set about creating various gating networks that were trained using the outputs of various combinations of these experts, while the weights of the trained expert networks themselves remained unchanged. Not only does this approach possess some of the advantages of committees of networks and mixtures of experts, in that it permits independent trends in different subsets of the data to be independently utilized, but it also provides us with an opportunity to scrutinize the usefulness of each of our data subsets alone. In the next subsection, we describe our experiences while constructing each of the ten subnetworks. We close the section by showing the results of using all these networks together, as well as in selected combinations.

5.3.2 Results with Individual Experts

Word-Length-Frequency Expert

Since we had had some success with backpropagation with weight decay in the previous experiments, our first network, containing 12 hidden units, used this algorithm. We quickly abandoned this approach after receiving training and test MSE's of around 0.216, even after training this network with several thousand cycles. Switching to backpropagation with momentum, with the learning rate set to 0.2 and momentum to 0.1 yielded the highly encouraging training MSE of 0.1839, test MSE of 0.1770. Unfortunately, further training of this network yielded the classic symptoms of overfitting—the training error decreased slightly and the test error increased sharply. Even these encouraging results came with a caveat: the network's outputs tended only to vary from 0.14 to 0.33, even though the magnitudes of all weights were less than 100.

Increasing the number of hidden units to 18 elicited a network with training MSE at 0.184 and test MSE of 0.178. Though it might have been highly useful to have decreased the size of the hidden layer, particularly in light of the results we present below for time-delay networks, we were convinced by the tendency of the networks we had tested to overfit or stall after reaching test MSE's of around 0.178 that no further experiments

Num. Hidden Units	Num. Cycles	Training MSE	Test MSE
8	2000	0.1842	0.176
	3000	0.1839	0.177
	4000	0.1839	0.1779
11	700	0.186	–
6	2000	0.1842	0.1764
8	3000	0.1833	0.1763
	4000	0.1832	0.1764
	8000	0.1831	0.177
	11000	0.1832	0.1834

Table 5.1: Results obtained for various networks used to test syllable frequencies. The table is given in chronological order of testing, and each set of values bounded by horizontal bars represents the results for a single network; a dash represents a value that was not recorded.

with our word-length data were needed. For the purposes of the combinations we describe below, we used a word-length expert with a test MSE around 0.1832 since its training MSE was very low and we were not readily able to recreate a network with a smaller test MSE at that time.

Syllable-Frequency Expert

A priori, we had hoped for somewhat better results from our syllable-frequency statistics than from our word-length statistics, so we used more combinations of tests on this network. We summarize our results in table 5.1. Since we had had reasonable results with the backpropagation with momentum approach in the word-length expert, we used it unchanged throughout these evaluations; i.e. the learning rate set at 0.2, momentum to 0.1.

As can be seen from table 5.1, the results we obtained were certainly good, compared with those for the amorphous networks, but were not significantly better than for the word-length expert. However, the distribution of values output by the network was much more satisfactory, ranging between 0.007 and 0.51. Also, weights within the networks continued to be relatively small, even in those with extremely long training sessions. For the purposes of the expert committees, we regenerated the 8-unit network with 8000 iterations of training noted in the last element of the table.

Average Sentence-Length Expert

In spite of our skepticism as to its usefulness, we expended considerable effort attempting to design a network to use sentence lengths at all effectively. The results of these experiments are presented in table 5.2.

Since backpropagation with momentum had proven reasonably effective in the past, we used it first again in this series of experiments. As table 5.2 makes clear, its results were hardly spectacular in this case; in spite of varying the number of hidden units and permitting incredibly long training sessions, these results are considerably worse than for either of the two previous tests. More disturbing was the fact that, upon examination, we discovered that in the first round of tests, whatever the network size the results were almost constant regardless of the data. Reducing momentum to 0.05 did not help: the value produced by the network simply got smaller—a result easily reducible to a different starting configuration. Weight decay certainly was not the answer; the table makes it plain how ineffective this algorithm proved. So we were obliged to halt our examination of this type of network, and conclude that sentence length averages are indeed a poor predictor of authorship contribution boundaries; for the purposes of future expert committees that would incorporate these statistics, we regenerated the network corresponding to the first line in the table.

Learning Algorithm	Num. of Hidden Units	Cycles	Training MSE	Test MSE
Backpropagation with Momentum	4	2500	0.1892	0.1821
	4	2500	0.1893	0.1814
	6	2500	0.1893	0.183
	4	5000	0.1893	0.182
Backpropagation with Momentum	4	2500	0.1894	0.1823
Backpropagation with Weight Decay	4	5000	0.222	0.214
		15000	0.222	0.220

Table 5.2: The results we obtained with networks trained on the average sentence length statistics we computed for each paragraph.

Each line in the table represents the results obtained with a particular network; where the number of hidden units is not given, training was continued on the same network as in the preceding line, and the number of cycles is cumulative. Each section bounded by horizontal lines represents the results achieved with a constant set of parameters used in a particular learning algorithm. As in table 5.1, dashes represent values that were not recorded.

Num. Hidden Units	Num. Cycles	Training MSE	Test MSE
30	2500	0.1363	0.175
	1250	0.1435	0.1722
	200	0.1570	0.1684
	100	0.1612	0.1638
25	100	0.1612	0.1740
35	100	0.1612	0.1658
	10000	0.1254	0.2020

Table 5.3: Results obtained for various networks used to test part-of-speech frequencies. The table is given in chronological order of testing, and each set of values bounded by horizontal bars represents the results for a single network. Throughout testing we used backpropagation with momentum, learning rate set to 0.2, momentum to 0.1.

Part-of-Speech-Frequency Expert

As disappointing as the results have proved to be for average sentence lengths, the same can certainly not be said for part-of-speech frequencies. As table 5.3 demonstrates, even the poorest results we observed with this category of statistic are superior to the best results we had yet observed.

The first and most heartening point about these results is that they demonstrate clearly and for the first time that we have created a network which can model the training set very well. We view this as important because, in all previous experiments, even when we experimented using ridiculously long training sessions, we were unable to get MSE's significantly lower than our best baseline; thus, concern had to be raised about whether neural nets could even in principle model the data (i.e. the data appeared to be very nearly random). Secondly, the best test results are over 10% better than the best baseline, and considerably better than any we had previously seen. Finally, even on test data the network output a very wide range of values—all the way from 0.00 to 0.81.

We cannot explain why three different networks trained to exactly the same level after 100 cycles. We also cannot deny that to experiment with multiple different starting positions for any particular topology would have been highly interesting. However, since we had achieved a very encouraging result as it was, we thought it best to press on with our investigation and return if we observed no better results.

Function-Word-Frequency Expert

Results obtained with function words were also encouraging. We made only two tests with function words, both using the familiar backpropagation with momentum learning algorithm with the accustomed parameters. The first attempt was with a network with 25 hidden units, which elicited a training MSE of 0.1720 and a test MSE of 0.1725 after 1250 cycles, and considerable overfitting thereafter. To improve generalization, we also tried to use a network with only 20 units; this produced a training MSE of 0.1728 and a test MSE of 0.1731. Taking these results as a positive indication, and planning to do more testing if we observed no better results, we moved on to the next data subset, using the 25-unit network we had first generated as one of the subnets in our expert committees.

Punctuation-Frequency Expert

As with our examination of function words, we did comparatively little testing with punctuation frequencies. Indeed, since our first guess at the appropriate size of the hidden layer had proven reasonable in the previous two experiments, we did not even experiment with this parameter; here we simply initialized a single network with 16 hidden units and viewed its performance on training sessions of various lengths. Our results, obtained again using backpropagation with momentum, are shown in table 5.4. We use network with the lowest test MSE in the expert committees we describe below; the performance of this network was quite good enough for this purpose.

Num. Cycles	Training MSE	Test MSE
2000	0.1565	0.1708
1000	0.1580	0.1734
500	0.1607	0.1684
250	0.1636	0.1693
150	0.1659	0.1664

Table 5.4: Results obtained for various networks used to test punctuation frequencies. The table is given in chronological order of testing. The fall in test error between 1000 and 2000 cycles appears to be anomalous; the general trend is in the reverse direction.

Entropy Expert

Entropy proved to be a far less fruitful field than the previous three data categories. While we did not experiment with many networks (only three different networks were used), it is difficult to imagine that our largest network, containing 32 hidden units, would not have been able to train to a better level than it did if such were possible; if there were any significant patterns in the data to be detected, this network should have at least hinted at them. Unlike the cases of word lengths and syllable frequencies, where the presence of an overall average could have biased the network (since the overall average has much greater magnitude than any of the frequencies), both entropy statistics habitually display similar values. Hence, these networks' poor performance is difficult to explain but by the fact that entropy, however measured, is a poor measure upon which to discriminate authors' contributions given small data sizes. Table 5.5 summarizes our findings. We used the first network we generated in subsequent experiments.

Vocabulary-Richness Expert

Of all the experts we had yet examined, in the few tests we conducted vocabulary richness proved the least effective. Our first test, using our standard backpropagation with

Num. Hidden Units	Num. Cycles	Training MSE	Test MSE
8	2000	0.1883	0.1779
16	2000	0.1884	0.1779
32	2000	0.1883	0.1781

Table 5.5: Results obtained for various networks used to test entropies. Again, backpropagation with momentum was used to train all networks.

momentum approach, produced a training MSE of 0.1929 and a test MSE of 0.1803 after 200 cycles, and very slightly worse results after 2000 cycles. Taking these results as indicative of what the backpropagation with momentum approach was capable of, and deciding, in view of our previous results, that adjusting the size of the hidden layer would likely not avail much (we set the hidden layers of these networks to 20 units throughout this trial), we decided to change the learning algorithm. Since backpropagation with weight decay seemed to offer nothing, and we needed an algorithm able to escape local minima with significant probability, we decided to try simulated annealing.

Annealing is the process by which materials, particularly metal or glass, are raised to a high temperature and then slowly cooled. From a physics standpoint, this treatment can make the material more regular because, when the temperature is raised, the individual molecules have substantial freedom to rearrange themselves. However, as the temperature is slowly decreased, they will tend to align in configurations requiring the least energy: as the temperature falls, the “energy threshold” a molecule has to overcome to move from one configuration to another rises, so that fewer and fewer stable structures have their configurations changed as time progresses, while less stable structures—with lower energy thresholds—become less frequent.

The analogy for neural networks is straightforward: at the beginning of the experiment, weights are randomly initialized. At any particular time step, a new weight configuration is randomly computed, and the error (in our case a squared error) is calculated. If it is less than the old error the new configuration is accepted; if it is greater, it will be accepted with a probability decreasing with time, according to an exponential distribution. Error and energy being treated as analogs, it is hoped that this procedure will result in a network with a low error level just as the physical process can result in a material with low energy.

Like all learning algorithms, this one has various parameters that seem to require experience to set properly. Chief among these are the minimum and maximum values which weights are permitted to attain. Setting these to -50 and 50 respectively yielded results that were worse than ridiculous (over 0.6 for training!) Setting them to -10 and 10 respectively produced the far more acceptable training MSE of 0.1889 and test MSE of 0.1814 after 10000 cycles and an appropriate setting of the parameters governing "temperature" and "cooling rate". Still the results are very poor, so we did no further testing with this data. We kept the last network for the purposes of our expert committees, since its outputs varied more than the network we had trained with backpropagation with momentum.

Hapax Expert

Table 5.6 summarizes the results we obtained while working with the hapax legomena and hapax dislegomena statistics that we computed. As the table shows, these results are better than those obtained with vocabulary richness statistics, but the improvement is not marked. It is particularly interesting to note how little in terms of training MSE the network benefited from an extremely long training session and a very large hidden layer; clearly this casts doubt on the usefulness of this type of statistic. For the purposes of our expert committees discussed in the next subsection, we kept the first network referred to in the table.

Learning Algorithm	Num. of Hidden Units	Cycles	Training MSE	Test MSE
Backpropagation	7	2000	0.1878	0.1781
with Momentum	40	10000	0.1877	0.1803
Simulated Annealing with Squared Error	7	10000	0.1881	0.1819

Table 5.6: The results we obtained with networks trained on the hapax statistics we computed for each paragraph.
Each line in the table represents the results obtained with a particular network. Each section bounded by horizontal lines represents the results achieved with a constant set of parameters used in a particular learning algorithm.

Waring-Herdan Expert

Having viewed the Waring-Herdan model as holding out considerable promise, we tested several networks trained with the statistics computed with it. The results of these experiments are presented in table 5.7, and as can be readily seen are not encouraging. Largely because the differences between the last network and the best are not significant, we used the last network in the expert committees we discuss in the next section.

Summary

Table 5.8 gives the best results we achieved with each of the ten data subsets. While we postpone a detailed discussion of the results until chapter 6, we note two interesting facts. First, all of our statistics performed better than any of the simple multilayer perceptrons we discussed in section 5.2. Second, the table clearly shows that our best results were derived from statistics measuring syntactic facets of a text rather than from those purporting to measure its lexical choice or to describe its author's vocabulary.

Learning Algorithm	Num. of Hidden Units	Cycles	Training MSE	Test MSE
Backpropagation	16	2000	0.1914	0.1801
with Momentum	16	–	0.1985	0.1868
Simulated Annealing with Squared Error	16	10000	0.1912	0.1798
Backpropagation with Momentum	16	5000	0.1911	0.1814

Table 5.7: The results we obtained with networks trained on the Waring-Herdan statistics we computed for each paragraph.
Each line in the table represents the results obtained with a particular network. Each section bounded by horizontal lines represents the results achieved with a constant set of parameters (except in the case of the second backpropagation with momentum network, for which we increased the momentum parameter from 0.1 to 0.5 to speed up learning) used in a particular learning algorithm. Dashes in the table represent values which were not recorded.

Expert Name	Number	Best MSE during Training	Best MSE during Test
Word-Length Frequencies	1	0.1836	0.1770
Syllable Frequencies	2	0.1831	0.1763
Average Sentence Length	3	0.1892;	0.1814
Part-of-Speech Frequencies	4	0.1612	0.1638
Function-Word Frequencies	5	0.1720	0.1725
Punctuation Frequencies	6	0.1659	0.1664
Entropy Statistics	7	0.1883	0.1779
Vocabulary richness	8	0.1929	0.1803
Hapax Statistics	9	0.1878	0.1781
Waring-Herdan Statistics	10	0.1912	0.1798

Table 5.8: The lowest test MSE's and corresponding training MSE's that were obtained with each of the ten "expert" networks that we trained on subsets of our data.

Num. of Hidden Units	Cycles	Training MSE	Test MSE
40	2000	0.1861	0.1781
*	5000	0.1867	0.1794
200	5000	0.1858	0.1774
	8500	0.1859	0.1794

Table 5.9: The results we obtained with committees comprising all ten networks we describe having trained in the last subsection, with a gating network of a certain size. The number of hidden units here refers to the number of hidden units present in the gating network. Each line in the table represents the results obtained with a particular network. The line with the * entry represents a network where training was continued with the learning rate increased from 0.2 to 0.3.

5.3.3 The Full Committee of Experts

We now turn to describing our first attempt to design a committee of experts to capitalize on the results discussed in the previous subsection. A priori, it seems to be reasonable to expect that it should be possible to design a gating network that would use each of the networks we trained in the previous subsection to achieve an accuracy at least as great as any of those component networks, and likely much greater. Unfortunately, as table 5.9 shows, this expectation may not have been realistic.

As described in subsection 5.3.1, we simply created a multilayer perceptron with ten input units, and connected this network with our pre-existing trained experts. We then trained this larger network, being careful to ensure that the experts' weights were frozen and that the gating network's weights had already been initialized.

We cannot but imagine that, given sufficiently many initializations, a configuration in the gating network would be produced that would lead to reasonable performance. That the networks we did create performed poorly seems to show that the dimensionality of the weight space is simply too large for the training data, and possibly also that information provided by each of the experts is often contradictory. We have resigned ourselves to the fact that using the whole of our data set is likely not useful; so, we spent much more effort in trying to create a "best" committee of experts using the three best experts as shown in subsection 5.3.4 below.

5.3.4 The "Best" Committee of Experts

Without question, the architecture from which we thought we would derive the best performance was the "best" committee of experts. Constructed exactly like the full committee of experts described in the previous subsection, except that it contained only the three most promising networks from subsection 5.3.2 (the part-of-speech-frequency expert, the function-word-frequency expert and the punctuation-frequency expert), this architecture seemed to get around most of the problems of contradictory information and weight-space dimension that were the primary explanations for the failure of the full committee. However, as table 5.10 shows, in spite of extensive testing this committee produced results only moderately better than did the full committee, bettering the performance of only one of its component networks.

The main division in table 5.10—represented by the double horizontal lines—occurs between experiments where the weights of the gating network were initialized between -1 and 1, and where they were initialized between 0 and 1. Clearly, the latter initialization is better because the experts should produce higher outputs with greater probability for contribution boundaries than for non-boundaries, so that it makes sense that there need be no inhibitory connections in the gating network. Indeed, were it not for the fact that we believe that using the full array of statistics is fundamentally not profitable, it might have been interesting to use this initialization on the full committee of experts.

Since we have used the scaled conjugate gradient, or SCG, algorithm to compute some of the results that we present in table 5.10, some explanation of the algorithm is

Learning Algorithm	Num. of Hidden Units	Cycles	Training MSE	Test MSE
Backpropagation with Momentum	50	5000	0.1852	0.1776
Simulated Annealing	50	10000	0.2021	0.1853
with Squared Error		13400	0.2519	0.2360
Backpropagation	50	200	0.1715	0.1680
with Momentum		2400	0.1791	0.1725
*	50	200	0.1885	0.1776
	50	200	0.1858	0.1779
**	50	5000	0.1844	0.1796
*	150	5000	0.1847	0.1793
SCG	50	4800	0.1870	0.1792

Table 5.10: The results we obtained with the "best" committee of experts. Each line in the table represents the results obtained with a particular network. Lines with no entry for hidden units refer to continuations in training of the network from the previous line, and the number of cycles is cumulative. Each section bounded by horizontal lines represents the results achieved with a constant set of parameters (except in the case of the the entries marked $*$, where the learning rate and momentum were set to $0.1, 0.05$ respectively, and the line labelled $**$, where those parameters were set to $0.04, 0.02$) used in a particular learning algorithm. The double horizontal line divides the first and second set of "correct" experiments.

Num. of Hidden Units	Cycles	Training MSE	Test MSE
50	550	0.1465	0.1605
50	1300	0.1380	0.1584

Table 5.11: The results we obtained with the "best" committee of experts when we failed to freeze the experts' internal weights.

in order. Whereas simple backpropagation relies only on the first derivative of the error with respect to weights to determine the direction in which the weight vector should be adjusted, SCG uses an approximation to the matrix of second derivatives plus a scaling factor, to make its estimate. This method was very successfully employed by Tweedie et al [33]. Its principal advantage is that it is guaranteed to reach the minimum of the error surface with regard to the initial starting position; the algorithm can also terminate automatically, unlike backpropagation methods which continue until stopped by some external criterion.

But, neither a more intelligent weight initialization scheme nor the use of a promising learning algorithm was able to raise this network's performance even to the level of two of its component networks. However, as so often happens in science, a simple error on the part of the experimenter led to a highly interesting result. While testing our network with backpropagation with momentum, we accidentally neglected to freeze the internal weights of the component experts. Since the results produced by this error were surprisingly promising, we conducted several further tests in which we made the mistake intentionally; the results are shown in table 5.11.

No doubt we should have performed more experiments, but since the second network in table 5.11 unequivocally produced better results than any of its component networks, we felt that our approach had been vindicated and that we could return to the problem if we saw no better results with our last architecture. We believe that the excellent performance of this network relative to the "properly" trained expert committee can be explained simply by the fact that the internal weights of the experts are reasonably well-adapted, but the network can further adjust them to make up for the poor (i.e. random) initialization of the weights in the gating network. Clearly, the gating network is useful, since this network outperforms any of its constituents; it seems the training simply allows the experts to be a bit more helpful, given the arbitrary position on the error surface from which the gating network begins training.

5.4 Time-Delay Neural Networks

5.4.1 Motivation

Notwithstanding the good performance of our best committee of experts discussed in the last subsection, we had one more network architecture we wanted to investigate. Time-delay neural networks are a comparatively recent invention: the SNNS documentation cites a paper written in 1989 as having been first to propose them. In certain problem domains, such as process control, one often has time-dependent data from which one must determine an optimal response. Since the data are time-dependent, it may be highly useful to look not only at the current set of data but also at the data produced in several preceding time intervals, because the patterns upon which the prediction should be based will be discernible only through the data's evolution over time. If the data sets are themselves ordered, so that each particular datum in a set is produced by the same process that generated that datum in previous time intervals, one could expect performance to be optimized if this homogeneity in the data could be incorporated in the predictive model.

Time-delay neural networks were conceived to use precisely this sort of data—that is, data that are time-dependent and for which corresponding data items from different sets measure the same underlying feature. Time-delay networks accomplish these tasks by grouping their input units into sets corresponding to each data set, so that the entire input layer has access to two or more data sets simultaneously. Weights for input units that are connected with the same feature in different data sets are coupled; the learning algorithm computes the derivative of the error function with respect to each weight in a standard fashion, but coupled weights are updated according to the average of the changes that would have been made to each weight were it independent. The term coupled should not be construed to mean that there are only two such weights; if data from two time intervals preceding the current one are being considered, then three coupled weights would exist between input units for each feature and each unit in the next layer. Other than having coupled weights, these networks may have arbitrary topologies, and may even have coupled units in their hidden layers which act in much the same way as in the input layer.

Intuitively, it is easy to cast our experiment to fit this model. Since the later paragraphs of a contribution must depend on earlier paragraphs, if we analogize time with sequence then there is a time-dependence to our data. Secondly, because our data sets are consistently ordered, we automatically have that each data item from one set measures the same underlying feature as the corresponding item from any preceding set. Therefore, it seemed very natural to us that time-delay networks represented a network architecture we should examine in detail.

Largely due to time considerations—training time-delay networks takes longer than training similarly-sized regular multilayer perceptrons because of the need to compute many averages—all of the time-delay networks we constructed have a fairly simple topology. All such networks only deal with two consecutive data sets. Further, at most one hidden layer is used, and no coupled weights are used in this hidden layer. Nonetheless, as we reveal below, we were able to obtain some highly interesting results.

5.4.2 The "Best" Time-Delay Network

Having already produced encouraging results, and also noting that this architecture seems never to have been applied previously in this field, we approached our investigation from the perspective of proving the null hypothesis—that is, that this architecture would produce results no better than those we had already observed. In this spirit, we decided to test it first on the three categories of statistics which, when combined in our "best" committee of experts, had given the best results we had seen. Table 5.12 shows that the null hypothesis was wrong, and that this type of network consistently produces superior results to any of those we have examined above.

The absence of variations of learning algorithms in the tests described in table 5.12 is simply due to the fact that SNNS only supports a variation of simple backpropagation for time-delay networks. For all of the networks described in the table, we used a very slow learning rate of 0.05, this having produced good results in our "best" committee of experts. We should also note that the network with 0 hidden units is a simple perceptron with a time-delay architecture.

Table 5.12 is interesting in many respects. Not only do most of the networks have better test results than any we have seen previously—the best over 5% better than the peak of our "best" committee of experts—but those results are very consistent, even arising with very small networks. Even a network with 2 hidden units performs quite well, although it required an absurdly long training session and its training error is still quite high. The performance of the 4-unit network is indeed quite acceptable. On the other hand, the simple perceptron performs quite poorly, worse even than our expert committees. This implies that the information the networks are using to make their predictions is nonlinear (simple perceptrons are known only to be able to make linear discriminations) but is also not nearly as extensive as we might have predicted. We treat these promising results further in the conclusion.

Num. of Hidden Units	Cycles	Training MSE	Test MSE
96	150	0.1405	0.1496
	950	0.0871	0.1722
72	150	0.1399	0.1515
	850	0.0945	0.1769
48	150	0.1412	0.1497
	300	0.1300	0.1543
32	150	0.1420	0.1500
	850	0.1053	0.1660
24	250	0.1350	0.1510
	400	0.1273	0.1564
20	225	0.1380	0.1495
	3000	0.0744	0.2107
16	100	0.1460	0.1506
	4225	0.0744	0.2231
12	175	0.1427	0.1501
	2600	0.1045	0.1832
8	275	0.1394	0.1499
	4350	0.1133	0.1831
6	275	0.1415	0.1501
	5000	0.1236	0.1701
4	275	0.1415	0.1501
	1850	0.1291	0.1615
2	4825	0.1473	0.1504
0	5000	0.1600	0.1619

Table 5.12: The results we obtained with the first time-delay neural network architecture that we tested.
As in the previous tables, each section bounded by horizontal bars represents the results achieved for a particular network, and each line where the number of hidden units is omitted indicates that training of the network of the previous line was continued, so that the number of cycles is cumulative.

5.4.3 Other Time-Delay Neural Networks

The success of the application of time-delay networks to our three best statistical categories made us curious as to their efficacy on other combinations of statistics, and particularly how they would compare with the more traditional network architectures we have already employed. We performed several experiments in this vein and we deal with them briefly in this subsection.

Our first attempt was to train a time-delay network using all of our statistical categories. After 5000 cycles, we found that the training MSE of this network was 0.1874 and its test MSE was 0.1804. While it is true that this network's learning rate was slow—we were still using the learning rate of 0.05 that had served us well throughout our other time-delay network trials—and our hidden layer was quite small, only 30 units, we did no further training on this network because progress, though apparent, was far too slow to make an interesting result likely. These results are indeed better than those we achieved with the simple multilayer perceptrons that we described in section 5.2, they are slightly inferior to the full committee of experts that we described in subsection 5.3.3.

We also decided to create time-delay networks for several individual categories of statistics. Still hoping to salvage some reasonable performance from our vocabulary richness statistics, they were the first we tested. This network's results were infinitesimally better than those of our best simple network: a training MSE of 0.1888 and a test MSE of 0.1806 after 5000 training cycles. The performance of the time-delay network we constructed to test our syllable frequencies was a more significant improvement over that of its predecessor: our lowest test MSE, observed after 1375 training cycles, was 0.1739 with a corresponding training MSE of 0.1832. The network we created for part-of-speech frequencies also performed better than any we had previously constructed for this subset of our data, attaining a test MSE of 0.1592 and a test MSE of 0.1544 after 925 cycles. Disturbingly, however, the network's training error had only decreased to 0.1478 after 5000 cycles, in spite of the network having 8 hidden units.

Feeling that we had exhibited a general, if not terribly significant, trend, we constructed no further networks. The results presented in this section do tend to support the claim that time-delay networks are generally better at predicting contribution boundaries than are simpler architectures, but for single categories the difference is not dramatic.

5.5 How Good is Our Best?

Up until this point, we have discussed our results mainly in relation to each other, and only using the abstract concept of mean squared error. To arrive at some sense of whether our results are good, we must compare them with some baseline tests. Using measures other than mean squared error to characterize our results might also give us some valuable insights. In this section we attempt to reach both these goals.

Three possible baseline tests have suggested themselves. The first, and simplest, is always to guess that there is no contribution boundary. This is motivated by the fact that the ratio of contribution boundaries to non-boundaries is almost exactly 1 to 3 in the entirety of our corpus; thus, an algorithm that always guesses 0 (no boundary) will produce a comparatively good mean squared error. Another possible baseline test would be to use this ratio of breaks to non-breaks to calculate the mean squared error for an algorithm which always guesses a single, optimal value. A third option is to design a simple algorithm that guesses 1 and 0 with appropriate probabilities, but makes its guesses randomly.

The first of these tests results in a mean squared error of 0.2552 for our entire corpus, and 0.2359 on the test set alone. Plainly, all of our networks perform substantially better than this—even those which output very nearly constant values independent of their input. This test does serve to highlight the difference between our test set and the overall corpus: our test set is substantially "easier" because there are proportionally fewer contribution boundaries. Whereas the entire corpus is almost exactly 1 to 3, the test set's ratio is just above 3 to 10. This fact should address the concern the careful reader will have been pondering for most of this chapter, namely why the performance of most of our poorer networks on training data has been considerably worse than on test data.

To calculate the single number that will produce the optimal MSE, we simply minimize the expression

$$\frac{r(1-x)^2 + wx^2}{r+w}$$

where r is the number of contribution boundaries and w the number of non-boundaries, with respect to x. This produces the general result

$$x = \frac{r}{r+w}$$

so that $x = 0.2552$ for the entire corpus and $x = 0.2359$ in the test corpus. This easily leads to an optimal MSE of 0.1901 on the entire corpus and 0.1801 on the test corpus. While these differ in magnitude by only 5%, the optimal test MSE is very close to the results obtained for all of the networks trained on all data, as well as to several data categories. It is plain that this result calls into question the significance of all but our very best networks.

Neither of these tests is amenable to producing a wider range of statistics than MSE's. For instance, since each is constant valued, it is not meaningful to compute precision, recall, or fallout statistics. It is more interesting to compute these sorts of statistics for our third baseline test, even though its MSE tends to be very high, usually around 0.37. Table 5.13 gives the results we obtained with one particular run of this test.

While the statistics presented in this table are a trifle nonstandard, they do illustrate that we got about 1/4 of the boundaries there were to get, and about 1/4 of what we got were actually contribution boundaries (and similarly for non-boundaries), precisely as we would have expected. In contrast to these results, we present a table of selected values

	Percent Correct	Percent Wrong
Contribution Boundaries	25.68	73.75
Non-Contribution Boundaries	75.29	25.28
Total:	62.62	37.38

Table 5.13: Results computed on one particular run of the third baseline test. The results are characteristic of all other runs we have made. The percentages of correctly-identified entities are relative to the total number to be identified (recall in the top row) and the incorrect column percentages are relative to the total guesses made.

generated from the very best network, from an MSE perspective, that we produced—the 20-hidden unit time-delay network operating on the three best statistical categories. These results are shown in table 5.14.

Several things about this table are noteworthy. First, we are able, simultaneously, to achieve better than 50% precision and recall. Table 5.13 would seem to indicate that such a result for a random process, even one having knowledge of the relative frequency of contribution boundaries, is highly improbable indeed. The accuracy of the random process was 62%, while that of our best network is nearly 80% at its peak. Clearly this is a very wide difference: indeed, even in the baseline that always guesses that no article boundary is present, the accuracy on the test set is only 76.4%, considerably lower than that of our best network.

We performed one more test to try to gain more confidence that our network is generating results that correlate with our data. Although neither the distribution of the outputs of our networks nor the distribution of the observed values is even close to normal, we decided to apply a standard t-test to these distributions to see if they correlated significantly. Somewhat surprisingly, the results we obtained were so strong they were far off the scale for a sample size of over 1000, implying that the distributions were extremely similar. It seems clear that this result is mainly attributable to the

Threshold	Precision	Recall	Fallout	Accuracy	Error
0.0500	0.2867	0.9191	0.2937	0.4413	0.5587
0.1000	0.3333	0.8211	0.4928	0.5703	0.4297
0.1500	0.3670	0.7402	0.6056	0.6374	0.3626
0.2000	0.4130	0.6691	0.7063	0.6975	0.3025
0.2500	0.4598	0.6029	0.7812	0.7392	0.2608
0.3000	0.5124	0.5564	0.8365	0.7704	0.2296
0.3100	0.5127	0.5441	0.8403	0.7704	0.2296
0.3200	0.5314	0.5392	0.8531	0.7791	0.2209
0.3300	0.5335	0.5270	0.8577	0.7796	0.2204
0.3400	0.5394	0.5196	0.8630	0.7820	0.2180
0.3500	0.5443	0.5123	0.8675	0.7837	0.2163
0.3600	0.5565	0.5074	0.8751	0.7883	0.2117
0.4000	0.5780	0.4632	0.8955	0.7935	0.2065
0.4500	0.5850	0.4216	0.9076	0.7929	0.2071
0.4600	0.5944	0.4167	0.9122	0.7953	0.2047
0.4700	0.6036	0.4069	0.9175	0.7970	0.2030
0.4800	0.6084	0.3922	0.9220	0.7970	0.2030
0.4900	0.6129	0.3725	0.9273	0.7964	0.2036
0.5000	0.6183	0.3652	0.9304	0.7970	0.2030
0.5100	0.6223	0.3554	0.9334	0.7970	0.2030
0.5200	0.6221	0.3309	0.9379	0.7947	0.2053
0.5300	0.6172	0.3162	0.9394	0.7924	0.2076
0.5400	0.6117	0.3088	0.9394	0.7906	0.2094
0.5500	0.6000	0.2941	0.9394	0.7872	0.2128
0.6000	0.6562	0.2574	0.9584	0.7929	0.2071
0.6500	0.6693	0.2083	0.9682	0.7889	0.2111
0.7000	0.7083	0.1667	0.9788	0.7872	0.2128
0.7500	0.7258	0.1103	0.9871	0.7802	0.2198
0.8400	0.8696	0.0490	0.9977	0.7739	0.2261
0.8800	0.9167	0.0270	0.9992	0.7698	0.2302
0.9000	0.8333	0.0123	0.9992	0.7663	0.2337
0.9200	1.0000	0.0074	0.0000	0.7658	0.2342

Table 5.14: Selected results computed by running our best time-delay network on our test suite.
The “threshold” is the value below which the network outputs are considered to be 0, above which they are taken to be 1.

fact that the conditions for the t-test's application were violated, but it does add at least infinitesimally to the body of evidence in support of the contention that our best networks are indeed capable of predicting the correct output with significant probability.

Chapter 6

Conclusions and Future Work

Having just set forth the results that we obtained from our various experiments in the preceding chapter, we now turn to the task of using them to draw some general conclusions. After examining the theoretical and practical questions upon which our results seem to shed light, we note the outcome of a concurrent experiment that tried to solve our problem using very different techniques. We close the chapter by pointing to some avenues for future research.

6.1 Conclusion

In this ambitious project, we have undertaken several different investigations. Firstly and most importantly, we have demonstrated that it is possible to design a system which, with probability significantly better than even an intelligent baseline test, can infer the presence of contribution boundaries using stylistic statistics, notwithstanding the fact that those statistics are computed over very small units of text. We have also shown that this is best done by statistics which capture a high-level element of style. Punctuation frequencies and part-of-speech frequencies, two of our best categories, appear to measure directly either an author's use of punctuation or his/her preferences of grammatical constructions. Function-word frequencies also proved reasonably effective and are thought to capture information concerning the phrase structures and modes of expression preferred by an author.

On the other hand, and probably just as usefully, we have shown that several categories of stylistic statistics seem to perform very poorly on short text lengths. Vocabulary richness, Waring-Herdan frequencies and hapax legomena and dislegomena all appear to offer virtually nothing. While this is hardly surprising for these measures—intuitively it is easy to imagine that vocabulary richness is only measurable with sufficient text to characterize a vocabulary—more surprising was the failure of our entropy statistics. Juola's measure in particular has been reported to work very well with samples only slightly larger than our average paragraph. It might be argued that our combination of Juola's measure with the admittedly unlikely lexical entropy may have destroyed the usefulness of Juola's statistic. Had we not chosen a neural net architecture, and tested these two statistics in isolation and with networks with large hidden layers, this argument might be sustainable; however, in light of our procedure we must conclude that Juola's statistic either is not useful for these purposes or simply requires more data.

Unsurprisingly, sentence-length statistics proved quite unhelpful. However, word-length frequencies and syllable frequencies showed at least a modicum of promise, particularly when a time-delay architecture was trained with them. This performance is particularly interesting considering that these tests are not at all well-designed, since we test both frequencies and overall averages simultaneously (hence the unnormalized averages must predominate over the very small frequencies).

Our results add force to the contention that neural networks are a useful tool in stylometry. While it certainly does not seem that simple multilayer perceptrons trained on amorphous sets of stylistic data are useful, we have exhibited several architectures which have produced quite impressive results. Committees of experts produced excellent results, particularly when the training of the gating network was combined with that of the individual pretrained experts. Along the way, we have shown that there is profit in combining categories of stylistic statistics together, if it is known in advance that those categories themselves perform quite well.

Perhaps our most original contribution has been the application of time-delay networks to our problem. We are not aware of any literature in stylometry in which this network architecture has been used. The fact that these networks have consistently proven to give better results than even complex multilayer perceptrons or committees of experts should serve as a beacon to others that this learning paradigm deserves considerably more attention than it has been accorded to date. Indeed, in any enterprise where a document is being sequentially examined—such as to discover redundant information in a document assembled by automatic information-retrieval—it is not at all difficult to imagine that these networks could prove to be a singularly valuable tool.

It is true that even our best results are never able to recover much more than 50% of the contribution boundaries with much better precision than 50%. What is far less clear is how well any technique relying strictly on whatever stylistic information could ever perform. Due to the high proportion of articles by a small number of contributors (especially the list's moderator) to `comp.risks`, it is very likely that a significant number of adjacent contributions share the same author. Recalling the genre- and domain-restrictions inherent in the corpus, it is also not at all difficult to envision instances in which the styles of contributions by separate authors are simply not distinguishable even by a human expert. With these factors in mind, it seems quite conceivable that we might have found far more than 50% of the boundaries that could in principle be discovered. Since we cannot rule out intra-sample discrepancies, such as those created by unidentified quotations, it is reasonable to conjecture that our precision may be considerably better than 50% as well.

Whatever importance one attaches to these hypothetical or unquantified factors, in view of the difficulty of the problem and the multiple directions in which we have broken new ground, it seems fair to claim that our project was successful. Had our data, particularly our part-of-speech frequencies, been more reliable, our results might be expected to have been much better. Even so, we contend that we have made both practical and theoretical advances, and that our work could serve as a cornerstone for a system to make simpler the jobs of those people who must assemble vast, collaboratively-written documents.

6.2 Related Work

Concurrent with this work has been a series of experiments undertaken by Bhaskara Marthi. These experiments utilize the data we produced in the first phase of this study, and compute from them vectors composed of letter-bigram frequencies.

Based on some work published in 1994 that applied bigram frequencies to the Federalist Papers, the goal of this study was to determine whether this technique could be applied to small text samples. The experiment worked by computing letter-bigram frequencies for each sample and using a simple cosine measure to determine the distance between two consecutive samples' vectors.

Unfortunately, despite some interesting attempts at weighting vectors according to the size of the samples from which they were derived, the results do not appear to be significantly better than random. For instance, it is never possible to obtain precisions higher than 0.6, and for most threshold values the precision is around 0.25, as one would predict for a random guess. Using a technique wherein recall is plotted against one minus fallout (we define fallout as the ratio of correct identifications of non-boundaries to the total number of non-boundaries), it can be shown that the results do not differ significantly from chance. Thus, in spite of the creative experiments, it does not seem that letter bigrams capture sufficient stylistic information in such small samples to make them useful.

6.3 Future Directions

Given the results we have obtained, several obvious lines of research seem to offer interesting possibilities. Chief among them is that leading further along the primary path this research was to carve out: it now seems that an attempt to develop a system to advise humans on the creation of stylistically-homogeneous documents is potentially feasible. Many hurdles need to be overcome, of course, not least exactly how statistical information about stylistic inconsistencies could be shaped into a human-comprehensible form. Also looming is the problem that neural networks are inherently extremely difficult to understand, and understanding the workings of a successful network would seem to be a prerequisite for designing such an interactive system—no other avenues of finding stylistic inconsistencies than that involving neural nets appear to promise comparable success.

One small step along this long road might be to examine our most successful networks in detail. This could be done both with a view to determining which statistics, or statistical subsets, seem to be most significant and also to attempting to find patterns within and between contributions that are treated well by the network. Both these tasks are extremely arduous, but if such patterns in the data or the network could be determined it might well be a critical advance for the design of interactive software.

A more mundane task would be to conduct further experiments with the data we have used here. Particularly with regard to the three most promising statistical categories, considerable insight might be had by testing various subsets of these categories together and independently. For example, it might be useful to know whether low-frequency function words help, or whether certain punctuation marks or part of speech tags can safely be ignored. Other methods of predicting contribution boundaries from the statistics could also be tried, such as correspondence analysis, rule induction, decision trees or even genetic algorithms. It might also be worthwhile to see whether windows larger than two parts might prove useful, particularly in the context of time-delay networks. We avoided this topic because it is too computationally intensive to be attempted within our limited time; however, it might prove highly instructive to experiment with varying window sizes. Clearly, a user-oriented system would have to be vastly more efficient and robust than our multi-phase procedure, and this too would necessitate considerable development, if little new research.

Setting aside all these technical issues, several more fundamental questions still require an answer. Glover and Hirst [5] have indeed shown that humans can use authors' styles to differentiate collaboratively-written works from those with a single author. Yet, their work gives hardly any insight into how this is done: just what characteristics of the text of a document do humans associate with stylistic inconsistency? It may well be that the cues our algorithm uses to identify stylistic inconsistency would be overlooked by a human reader, while factors which would detract markedly from a reader's comprehension or enjoyment of a work pass undetected by our method. Certainly, these questions would have to be answered by any effort intent on designing a usable style advisor; their answers, though, would be very interesting in their own right.

A distinct set of questions concerns whether human perceptions of stylistic inconsistency are so subjective that no general algorithm to detect such inconsistencies could ever be developed. Baljko and Hirst [1] have investigated these questions by asking a number of subjects to read a set of paragraphs, all written on a difficult philosophical problem and produced by several different authors, and then independently to organize them into stylistically-similar groups. They show that there is a significant correlation amongst the resultant groupings, but that the groupings do not always correlate with authorship. Even though great care was taken, through choosing the domain of discourse of the writing so as to be highly esoteric to ensure that the content of the samples did not influence the humans' groupings, it seems very hard to believe that this could have succeeded completely. Nonetheless, this study does lend support to the idea that the concept of objective stylistic inconsistency does exist. However, we are inclined to doubt its conclusions regarding a lack of connection between authorship and stylistic inconsistency, both because of our concerns regarding the influence of content and because of the small size and domain specificity of the samples. Further research on this matter is imperative.

An experiment to contrast our algorithm's performance in recognizing stylistic inconsistencies with the perceptions of humans would also be very much in order. The difficulties inherent in such an experiment should not be underestimated, however: for instance, our corpus would not be usable for this purpose because fluctuations in style will inevitably coincide with changes in topic, focus, or perspective. While these content changes will not be sufficiently dramatic to influence our statistically-based methods, no human exists who has the mental discipline to consistently ignore them when reading for stylistic inconsistencies. Either the data used by Glover and Hirst or that from the Baljko and Hirst study might be somewhat more valuable, but in either case the corpus's small size and restricted domain would limit the general applicability of whatever conclusions were drawn.

What would be required is some set of reasonably large documents, each generally regarded as not being stylistically cohesive, which a group of humans could be set to read and note where they felt stylistic inconsistencies occurred, and possibly asked for their reasoning. While collaboratively-written documents are obvious candidates for such work, single authors are quite capable of producing stylistic inconsistencies—as anyone who has tried to write a long document over an extended period will doubtless attest—so that documents written by one person but containing apparent inconsistencies could also be considered. Nor does it seem necessary that a "right" answer exist—that is, that some set of style boundaries should be recognized in advance. Indeed, such right answers, like those in our study, not only will often coincide with content changes that human readers will unavoidably perceive, but may also not be right at all—some authors' styles, especially in such a constrained domain as ours, may simply not be distinguishable by any means.

Whether the human readers were required to provide reasoning or not, undoubtedly the most interesting aspect of this experiment's results would be to investigate how well the humans' identifications clustered and correlated with the output of ours or a like algorithm. A wide degree of disagreement among humans would run counter to the results of Baljko and Hirst, and suggest that the problem of identifying stylistic inconsistencies is intrinsically subjective, and thus the best computer algorithm could never be expected to do more than suggest areas of possible inconsistency. Disagreement between humans and the algorithm would indicate that the techniques used by our algorithm to detect auctorial changes, while successful for this purpose, are not related to humans' perception of style. Close correlation between the humans and the computer, or general correlation with exceptions, would be the most favorable result, and would allow for an examination of precisely what sorts of inconsistencies are identified by what statistical categories as well as how inconsistencies might be described. In any event, such a study would give an idea of an upper bound on the performance to be hoped for in any algorithm—an upper bound that we do not currently possess.

Turning to the field of learning techniques, we would be remiss not to take this opportunity to strongly advise more work with time-delay networks. This novel architecture could have applications in many fields of natural language processing in addition to stylometry, and their impressive performance in this study certainly appears to merit additional study. Indeed, complex multilayer perceptrons like those employed in our committees of experts show promise themselves, and should also be examined further.

Finally, this study seems to throw strong support behind the assertion that, particularly for small text samples, high-level statistics hold far more information than lower-level statistics. If part of speech tags perform so well, it must be asked whether even higher-level statistics such as frequencies of noun phrases of various lengths, proportion of prepositional phrases, etc. might not be even more useful. Both Stamatatos et al [27] and Hatzivassiloglou et al [6] have successfully demonstrated applications for these sorts of statistics, and a study linking these ideas with powerful neural net architectures might prove very interesting. Whatever may eventually become of the idea of an automatic style advisor, it is apparent that this thesis has opened the door to many potentially fruitful veins of research.

Bibliography

[1] Baljko, Melanie and Hirst, Graeme, "Subjectivity in Stylistic Assessment", *Text Technology* (to appear, 2000).

[2] Battelle Memorial Institute, Pacific Northwest National Laboratory, "Neural Networks: Freeware and Shareware Tools" (Available at: `http://www.emsl.pnl.gov:2080/proj/neuron/web/neural/systems/-shareware.html`).

[3] Bishop, Christopher M., *Neural Networks for Pattern Recognition* (Oxford University Press, 1995).

[4] Brill, Eric, "Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging", *Computational Linguistics*, 21(4), 1995, 543-565.

[5] Glover, Angela and Hirst, Graeme, "Detecting Stylistic Inconsistencies in Collaborative Writing", in Sharples, Mike and van der Geest, Thea, Eds., *The New Writing Environment: Writers at Work in a World of Technology* (Springer-Verlag, 1996), 147-168.

[6] Hatzivassiloglou, Vasileios, Klavans, Judith L. and Eskin, Eleazar, "Detecting Text Similarity over Short Passages: Exploring Linguistic Feature Combinations via Machine Learning", *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999, 203-212.

[7] Holmes, David I., "Authorship Attribution", *Computers and the Humanities*, 28(2), April 1994, 87-106.

[8] Holmes, David I. and Forsyth, Richard S., "The *Federalist* Revisited: New Directions in Authorship Attribution", *Literary and Linguistic Computing*, 10, 1995, 111-127.

[9] Jacobs, R.A., Jordan, M.I., Nowlan, S.E., and Hinton, G.E., "Adaptive Mixtures of Local Experts", *Neural Computation*, 3(1), 1991.

[10] Juola, Patrick, "Cross-Entropy and Linguistic Typology", *Proceedings of New Methods in Language Processing 3*, Sydney, Australia, 1998.

[11] Juola, Patrick, "What Can We Do with Small Corpora? Document Categorization Via Cross-Entropy", *Proceedings of an Interdisciplinary Workshop on Similarity and Categorization*, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK, 1997.

[12] Karlgren, Jussi, "Stylistic Variation in an Information Retrieval Experiment", *Proceedings of New Methods in Language Processing 2*, 1996.

[13] Karlgren, Jussi, and Cutting, Douglas, "Recognizing Text Genres with Simple Metrics Using Discriminant Analysis", *Proceedings of the 15th International Conference on Computational Linguistics (COLING 94)*, Kyoto, 1994, 1071-1075.

[14] Kessler, Brett, Nunberg, Geoffrey, and Schütze, Hinrich, "Automatic Detection of Text Genre", *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the Association for Computational Linguistics*, Madrid, 1997, 32-38.

[15] Kučera, Henry and Francis, W. Nelson, *Computational Analysis of Present-Day American English* (Brown University Press, Providence, Rhode Island, 1967).

[16] Lancashire, Ian, "Probing Shakespeare's Idiolect in Troilus and Cressida", *University of Toronto Quarterly*, 68(3), Summer 1999, 728-67.

[17] Levin, Beth, *English Verb Classes and Alternations: A Preliminary Investigation* (University of Chicago Press, 1993).

[18] Manning, Christopher and Schütze, Hinrich, *Foundations of Statistical Natural Language Processing* (The MIT Press, Cambridge, MA, 1999).

[19] Mason, Oliver, "QTAG", (Birmingham University, 1998).

[20] Matthews, Robert A. J. and Merriam, Thomas V. N., "Distinguishing Literary Styles Using Neural Networks", in Fiesler, Emile and Beale, Russell, Eds., *Handbook of Neural Computation* (CD-ROM Edition, Oxford University Press, Release 97/1, Section G8.1).

[21] Mealand, David L., "Measuring Genre Differences in Mark with Correspondence Analysis", *Literary and Linguistic Computing*, 12(4), 1997, 227-245.

[22] Mitton, Roger, "Spelling Checkers, Spelling Correctors and the Misspellings of Poor Spellers", *Information Processing and Management*, 23(5), 1987, 495-505.

[23] Mosteller, Frederick and Wallace, David L., *Inference and Disputed Authorship: The Federalist*, (Addison-Wesley Publishing Company, Inc., Reading, MA., 1964).

[24] Ratnaparkhi, Adwait, "A Maximum Entropy Part-Of-Speech Tagger", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, May, 1996, University of Pennsylvania.

[25] Santorini, Beatrice, "Part-of-Speech Tagging Guidelines for the Penn Treebank Project" (University of Pennsylvania, 1995).

[26] Schmid, Helmut, "TreeTagger—a Language Independent Part-of-speech Tagger", (Institut für Maschinelle Sprachverarbeitung (IMS) Universität Stuttgart, 1995).

[27] Stamatatos, Efstathios, Fakotakis, Nikos and Kokkinakis, George, "Automatic Authorship Attribution", *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, June 1999, 158-164.

[28] Stamatatos, Efstathios, Michos, S., Fakotakis, Nikos and Kokkinakis, George, "A User-Assisted Business Letter Generator Dealing with Text's Stylistic Variations", *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence*, 1997.

[29] "Stuttgart Neural Network Simulator", University of Stuttgart, Institute for Parallel and Distributed High Performance Systems (IPVR), Stuttgart, Fed. Rep. of Germany, 1990-95, and University of Tübingen, Wilhelm-Schickard Institute for Computer Science, Tübingen, Germany, 1996-98.

[30] Teahan, William J., *Modelling English Text*, (Ph.D. Thesis, Department of Computer Science, Waikato University, 1997).

[31] Thede, Scott M., *Parsing and Tagging Sentences Containing Lexically Ambiguous and Unknown Tokens* (Ph.D. Thesis, Purdue University, August 1999).

[32] Tweedie, Fiona J., Singh, Sameer and Holmes, David I., "An Introduction to Neural Networks in Stylometry", in *Research in Humanities Computing*, 5, (Oxford University Press, 1996), 249-263.

[33] Tweedie, Fiona, Singh, Sameer and Holmes, David, "Neural Network Applications in Stylometry: The *Federalist Papers*", *Computers and the Humanities*, 39(1), 1996, 1-10.

[34] van Halteren, Hans, Zavrel, Jakub, and Daelemans, Walter, "Improving Data Driven Wordclass Tagging by System Combination", *Proceedings of the 36th Annual Meeting of the Association For Computational Linguistics and the 17th Annual Conference on Computational Linguistics, COLING/ACL*, Montreal, 1998, 491-497.

[35] Wilson, Michael, *MRC Psycholinguistic Database: Machine Usable Dictionary, Version 2.00* (Informatics Division, Science and Engineering Research Council, Rutherford Appleton Laboratory, 1987).