Dunedin, New Zealand, Springer-Verlag (1997) 865-868

A Fuzzy Logic Approach to Computer Software Source Code Authorship Analysis

R. I. Kilgour, A. R. Gray, P. J. Sallis and S. G. MacDonell

Department of Information Science, University of Otago, PO Box 56, Dunedin, New Zealand
Email: rikilgour@commerce.otago.ac.nz

Abstract

Software source code authorship analysis has become an important area in recent years with promising applications in both the legal sector (such as proof of ownership and software forensics) and the education sector (such as plagiarism detection and assessing style). Authorship analysis encompasses the sub-areas of author discrimination, author characterization, and similarity detection (also referred to as plagiarism detection). While a large number of metrics have been proposed for this task, many borrowed or adapted from the area of computational linguistics, there is a difficulty with capturing certain types of information in terms of quantitative measurement. Here it is proposed that existing numerical metrics should be supplemented with fuzzy-logic linguistic variables to capture more subjective elements of authorship, such as the degree to which comments match the actual source code's behavior. These variables avoid the need for complex and subjective rules, replacing these with an expert's judgement. Fuzzy-logic models may also help to overcome problems with small data sets for calibrating such models. Using authorship discrimination as a test case, the utility of objective and fuzzy measures, singularly and in combination, is assessed as well as the consistency of the measures between counters.

1 Introduction

Software metrics in the *traditional* sense most commonly refers to quantitative assessments applied to products or processes. Thus the number of lines of code in a series of programs, defect density, and length of documentation in pages could be considered to be fairly typical examples. Here, however, the term refers to the more specialized sub-field that has approached that task of measuring aspects of source code that relate in some manner or other to aspects of the programmer.

These relationships are assumed to exist due to the psychological makeup of the programmer, including the manner in which they approach the problem-solving process, the manner of programming to which they are accustomed, and various demographic variables such as experience and gender. In this manner, the task of software source code authorship analysis parallels to some degree written text authorship analysis [Sallis, 1994]. For example, attributing authorship of a new work to Shakespeare or assessing the psychological characteristics of a suspect as expressed in samples of writing in a forensic investigation. For this reason there has been considerable transfer of ideas and techniques from the traditional textual analysis and forensics fields to source code analysis. In the remainder of the paper references to authorship analysis refer specifically to source code analysis unless explicitly stated otherwise.

The field of authorship analysis can be conveniently partitioned into particular application goals, with some techniques and measures more appropriate for some goals than others. The three main areas where software metrics have been used for authorship analysis are author discrimination, author characterization, and similarity detection.

Author discrimination usually refers to the development of models that can identify which of a known set of authors was most likely to have written a new piece of source code. This is carried out by tuning the model with as many pieces of code from each author as possible. A related area is determining whether a given author could have written a piece of source code. This particular case overlaps with similarity detection, which is discussed below.

Author characterization is the task of associating personality and background factors of the author to features found in the source code. For example, the initial language that the author programmed in may affect how they program in subsequent languages.

The area of similarity detection has been the primary area of research given its implications for plagiarism detection in educational institutions [Jankowitz, 1988; Whale 1990]. This area overlaps substantially with authorship discrimination in some cases, but also exhibits its own unique problems and requirements.

In all of these cases, metrics that are used for developing these models are currently quantitative or categorical and are usually counted with an automated system, although hand-counting is also used in some cases. The principal problem with this approach has been the requirement that the metrics can be defined in an objective manner with an associated counting algorithm. This is trivial for many metrics, but is problematic for others since many of the more interesting metrics, such as how well the code and comments correspond to each other and spelling errors consistently made, are difficult (if not impossible) to quantify in a reasonable manner.

Here the approach taken is to use a combination of fuzzy-logic linguistic variables for subjective aspects, alongside the already developed numerical measures. Section 2 provides some more detail about software metrics for authorship analysis, Section 3 discusses the use of fuzzy logic metrics, Section 4 presents a small case study that illustrates the use of such hybrid models, and Section 5 considers the implications of this work and some areas which require further attention.

2 Software Metrics for Authorship Analysis

Although programming languages are admittedly less free form than natural languages, in terms of syntax and grammar, they still allow a certain amount of flexibility that enables programmers to *express* themselves in different ways. For instance, a given programmer may prefer a particular looping control construct over another functionally equivalent set of statements; or another programmer may be meticulous in terms of maintaining program nesting depths when compared to a colleague. There is substantial literature that details some of the many metrics used for this purpose and the interested reader is referred to [Berry and Meekings, 1985; Jankowitz, 1988; Spafford and Weeber, 1993] as useful summaries of some of these.

Our approach to authorship analysis is based on the construction of an *author profile* [Sallis *et al.* 1996], using a comprehensive set of program metrics. If programmers do indeed adopt particular styles in their coding then this should be evident in the constructs they use, and should therefore be measurable. The profile for a given programmer is likely to include metrics relating to product size, structure, layout, and expression. It should also incorporate some consideration of language analysis, in that a particular programmer may use a standard approach to naming variables, for example. *temp1*, *temp2*; or *C_key* and *O_key* for customer and order key respectively.

Measures of program size (both at the token and statement level) and structure are likely to be effective only in instances where a similar program (in terms of required functionality) written by the same author is available for reference. As unlikely as this may seem, it is common for programmers to retrieve and adapt pre-tested segments of code that they have written previously for use in a new system. After all, many systems within a given application domain will have components or modules in common (for example, an order processing system has a standard set of building blocks that will be customized as necessary). This is also likely to be the case within an education setting where plagiarism detection is the goal. If this is the case, then we could expect similar size and structure measures to be obtained from an analysis of the two or more sample modules. Indicators of layout (including nesting depth, white space, commenting, statement length) and syntactic expression, on the other hand, may be more generally associated with a given programmer, without requiring recourse to a functionally equivalent program. Thus levels of nesting depth and comment structure may be attributable to individual coders irrespective of the particular program they are developing.

Admittedly issues such as global code reuse from libraries, or the strict adherence to particularly detailed organizational standards could confound this type of analysis. Within an educational setting the degree of influence of the lecturer's code examples must also be considered, as well as the possible desirability of some collaboration between students. Having said this, however, there would still seem to be sufficient leeway for individual programmers to exhibit their own *brand* or style of coding in terms of comment content and structure and in the choice and use of variable, constant and label names [Spafford and Weeber, 1993]. It is our assertion, then, that a *comprehensive* profile that takes into account as many aspects of coding structure and content as possible, including characteristics of language use, should effectively enable the differentiation of program authors, given the availability of a sufficiently large pool of programs.

3 Fuzzy Logic Metrics for Authorship Characterization

The main advantage of using fuzzy variables is that they allow for the capture of concepts that programmers can identify with, such as complexity, deliberate versus non-deliberate spelling errors, and the degree to which code and comments match. It is suggested that expert programmers can classify code into fuzzy logic categories using such variables with relative ease and consistency.

Another advantages of using fuzzy logic is that by reducing the number of free parameters in the model, less data is required for calibration. For many applications of authorship analysis, the large quantities of data required for using neural network or statistical models are simply not available.

Finally, some metrics have been developed that attempt to avoid the problems inherent in inflexible counting algorithms and rely on the expert working through the programs applying subjective counting rules. Here it is suggested that single measures can be made with high levels of accuracy after a less stringent examination by using linguistic variables.

By developing a series of fuzzy logic metrics, fuzzy logic models can be created that combine these with traditional numerical measurements where this is appropriate. In this way the most useful set of metrics (a mixture of fuzzy and numerical variables) can be used together. Other fuzzy techniques such as fuzzy case-based reasoning could also be used.

The remainder of the paper will assume the use of the C++ programming language for examples. The concepts generalize to most other languages fairly readily. While the use of quantitative variables such as shown in Table 1 is still recommended it is suggested that they be supplemented with qualitative variables as shown in Table 2.

Table 1. Objective Measures

Metric	Objective Measures
1	Proportion of blank lines
2	Proportion of lines that are or include comments
3	Average length of identifiers
4	Use of templates
5	Statements per function/method
6	Use of underscores in identifiers
7	Use of capitalization in identifiers

Table 2. Fuzzy Variable Measures

Metric	Fuzzy Variable Measures
F1	Braces on separate lines
F2	Degree of indentation used
F3	Meaningful identifiers
F4	Use of utility variables
F5	Spelling errors
F6	Comments match code

Some of the variables shown in Table 2 could be quantified. For example the style of braces used could be classified into, say, opening on a single line, closing on a single line, both on a single line, and neither on a single line. Exceptions to the predominant rule could be counted and treated as proportions. However, the use of a fuzzy logic variable allows for a quick and easy measure to be taken, without unnecessary assumptions.

4 Illustrative Case Study

An illustrative experiment was carried out using a small amount of data. This experiment involved eight programs written in C++ by two textbook authors [Ammeraal, 1996; Flamig, 1995] who were also experienced software developers. The programs were implementations of quicksort, a generic text search, selection sort, and insertion sort algorithms (in that order). Metrics were extracted from the code, with a goal of discriminating between the two authors. This experiment was carried out as an illustrative pilot, as a full set of data is currently being gathered from a much wider range of programs and authors.

Firstly, some objective metrics (Table 1) were extracted from the code using a combination of an automated extraction tool and hand counting. Some analysis was then performed on these metrics. Secondly, the programs were presented to two experienced software developers for subjective analysis, and fuzzy logic metrics (Table 2) were derived using the best match of the labels in Table 3. After analysis of these fuzzy metrics for author discrimination, the metrics were combined to see if a synergy of the two forms could achieve better results. The results are shown in Table 4. Only one set of objective metrics (1-7) is shown for each program/author combination, along with the two experts' measurements of the fuzzy variables (F1-F6) in each case. Programs 1a to 4a represent the four programs written by Author A and programs 1b to 4b represent the corresponding programs written by Author B.

Table 3. Fuzzy Values

Fuzzy Value	Value in Table 4
Never/Almost Never	N
Occasionally	O
Sometimes	S
Most of the time	M
Always/Almost Always	A

Table 4. Data from Case Study

Metric	Prog 1a	Prog 2a	Prog 3a	Prog 4a	Prog 1b	Prog 2b	Prog 3b	Prog 4b
1	3/59	7/62	0 /12	0 /11	11/70	15/96	3/23	2/22
2	7/59	12/62	1/12	0/11	14/70	20/96	6/23	5/22
3	2.53	3.05	1.33	2.40	3.50	3.34	3.33	3.60
4	Y	N	Y	Y	Y	N	Y	Y
5	14.5	9.7	9.0	3.5	22.0	27.5	9.0	7.0
6	N	N	N	N	Y	Y	N	Y
7	Y	N	Y	Y	Y	Y	Y	Y
F1	O N	S O	O S	S S	M M	S O	M M	S S
F2	A M	M A	M M	O A	A A	M A	A A	M A
F3	S O	S S	S O	O N	M S	M S	S O	M M
F4	S O	M S	S S	M S	S O	M O	M S	M S
F5	N N	N N	N N	N N	O N	N N	N N	N N
F6	O O	S M	M S	N N	M M	A M	M A	A O

While this data is intended to be used purely as an illustration of how the two types of authorship measurements can be used together (since given the number of variables and observations the task of discrimination is trivial, and there is no hold-out validation set to confirm any relationships observed), some preliminary conclusions were tentatively drawn.

- Some programming tasks may suit certain styles (for example, the use of templates)
- Several programs may be needed to extract reliable metrics (for example, the use of underscores) where the behavior is not completely consistent

- Fuzzy metrics, although subjective, tend to be consistent over various analysts, especially after normalizing each expert's baseline. These metrics also allow for the collection of additional information not covered by the objective measures

The last point mentioned about the consistency of the expert's labeling of programs, providing that some form of normalization is used, is important to note. While it may in some cases be possible for the experts to discuss what constitutes a given level of some variable, and perhaps even identify prototype examples of programs that fit into each or some of the categories, the use of post-hoc rescaling appears to considerably reduce the differences between the measurements with minimal effort.

If this example was to be extended into a more complete model, then the creation of a fuzzy model, such as a fuzzy logic rule base, fuzzy case-based reasoning model, or fuzzy discriminant analysis model, would be carried out. This model would then use all available information, both fuzzy and numerical variables.

Again it is important to note that the above example is simply an illustration of how such fuzzy logic metrics could be defined and used for the task of authorship discrimination. The use of such metrics for authorship characterization and similarity detection is also seen as both feasible and desirable.

5 Conclusions and Further Work

It has been argued in this paper that such hybrid models, combining objective counts and classifications with fuzzy logic linguistic variables, not only improves the accuracy of the models, but also allows for faster counting over existing subjective schemes.

Given the necessity for both quantitative and qualitative measurements for authorship analysis applications, fuzzy-logic linguistic variables provide a promising approach to improving the accuracy and ease of use of such models. The objective and fuzzy variables can then be combined into a single model.

Current work is now focusing on developing actual models for authorship characterization, and further investigating the consistency of the fuzzy logic measures. A larger scale project is underway with the goal of collecting much larger quantities of data with a wider range of programs from a more diverse set of authors.

References

[Ammeraal, 1996] L. Ammeraal, *Algorithms and Data Structures in C++*, West Sussex, 1996.

[Berry and Meekings, 1985] R. Berry and B. Meekings, A Style Analysis of C Programs, *Communications of the ACM*, 28, 80-88, 1985.

[Flamig, 1995] B. Flamig, *Practical Algorithms in C++*, New York, 1995.

[Jankowitz, 1988]. H. Jankowitz, Detecting Plagiarism in Student Pascal Programs, *The Computer Journal*, 31, 1-8, 1988.

[Sallis, 1994] P. Sallis, Contemporary Computing Methods for the Authorship Characterisation Problem in Computational Linguistics, New Zealand Journal of Computing, 5, 85-95, 1994.

[Sallis *et al.*, 1996] P. Sallis, A. Aakjaer and S. MacDonell, Software Forensics: old methods for a new science, *SE:E&P'96*, 367-371, 1996.

[Spafford and Weeber, 1993] E. Spafford and S. Weeber, *Software Forensics: Can we track code to its authors?*, Computers & Security, 12, 585-595, 1993.

[Whale, 1990] G. Whale, Software Metrics and Plagiarism Detection, *Journal of Systems Software*, 13, 131-138, 1990.