On Improving Authorship Attribution of Source Code

Matthew F. Tennyson

Bradley University, Department of Computer Science & Information Systems, Peoria, IL, USA
mtennyson@bradley.edu

Abstract. Authorship attribution of source code is the task of deciding who wrote a program, given its source code. Applications include software forensics, plagiarism detection, and determining software ownership. A number of methods for the authorship attribution of source code have been proposed. This paper presents an overview and critique of the state of the art in the field. An independent comparative study is presented using an unprecedented experimental design and data set, as well as proposals for improvements and future work.

Keywords: authorship attribution, software forensics, plagiarism detection.

1 Introduction

In 1993, the term "software forensics" was coined to refer to the process of analyzing software – usually malicious remnants left after an attack – to identify the authors of the software in question or to at least identify characteristics of the authors [1]. The basic premise behind software forensics is that programmers generally apply a unique style to the code they write. As a result, programmers often leave "fingerprints" by embedding idiosyncratic features in their software. By identifying such features and associating them with a particular programmer, the original author of software whose author is otherwise unknown can be discovered.

The term "authorship attribution" refers simply to "the task of deciding who wrote a document" [2]. Features are generally analyzed regarding the style in which the document was written. These stylistic features might include the frequency or use of certain words, word length, word patterns, etc. Typically, documents of known authorship are used as training data, and the training results are then used to attribute an author to documents of unknown authorship. Numerous methods for authorship attribution have been proposed for natural language documents, including lexical methods, grammatical methods, and language-model methods.

Recent surveys of authorship attribution methods include those of Patrick Juola [3] and Efstathios Stamatatos [4]. Juola provides an historical context and analysis of some state-of-the-art methods in order to ultimately offer a recommendation for best practices. Stamatatos discusses the myriad applications of authorship attribution. An analysis of authorship attribution methods is also provided, which is focused on textual representation and computational requirements, providing a perspective grounded in information science.

Authorship attribution of source code, specifically, refers to the task of deciding who wrote a source code document. Authorship attribution is, therefore, a tenet of software forensics. Applications of source code authorship attribution include not only forensics investigations, but also plagiarism detection, software ownership disputes, and other similar activities.

Many researchers have contributed to the depth of knowledge regarding source code authorship attribution. Oman and Cook [5] were one of the first researchers to present a method for identifying authorship of programs based on programming style. Gray, Sallis, and MacDonell [6] introduced several metrics that can be used for authorship attribution of source code, and developed a tool called IDENTIFIED capable of extracting those metrics. Krsul and Spafford [7] performed one of the first in-depth studies of source code authorship attribution, analyzing several methods for determining authorship including discriminant analysis and several classification techniques using a tool called LNKnet. MacDonell, Gray, MacLennan, and Sallis [8] utilized neural networks, multiple-discriminant analysis, and case-based reasoning. Ding and Samadzadeh [9] utilized canonical discriminant analysis and 56 metrics to determine authorship. Lange and Mancoridis [10] utilized the similarity of histogram distributions of 18 code metrics, which were selected using a genetic algorithm. Elenbogen and Seliya [11] utilized a C4.5 decision tree. Frantzeskou, Stamatatos, Gritzalis, Chaski, and Howald [12] utilized Source Code Author Profiles (SCAP) using n-grams to represent programs and a similarity measure to determine authorship.

In 2010, Burrows [13] presented a comparative study that included most of the aforementioned methods of source code authorship attribution. The study consisted of a 10-class experiment (determining the author of a program from a set of ten candidate authors). A "leave-one-out cross validation" experimental design was used (each program in the data set was selected, in turn, as a query program while the remaining programs were used as training data). The results were measured in terms of accuracy (as a percentage of programs whose authors were correctly identified). The most effective method was found to be the Frantzeskou method [12].

In addition to presenting the comparative study, Burrows also presented a new method of source code authorship attribution. This new method was evaluated using the same 10-class experiment used to evaluate the other methods. The Burrows method performed the best on 3 out of 4 segments of the data set, while Frantzeskou performed the best on the remaining segment.

The Burrows and Frantzeskou methods are clearly state of the art in authorship attribution of source code. This paper presents an overview and critique of these methods, an independent comparative study of them using an unprecedented experimental design and data set, as well as proposals for improvements and future work.

2 Overview

Both the Frantzeskou and Burrows methods utilize n-grams to represent programs, and they both use a similarity measure to determine authorship. However, they are significantly different in the way the n-grams are formed and the specific similarity measures that are used. How the training programs are grouped is also a key difference. These differences will be delineated in the following sections.

2.1 The Frantzeskou Method

The Frantzeskou method uses source code author profiles to characterize the programs written by each author. The concept of author profiles is derived from the work of Keselj [14]. The concept of an author profile is defined as the set of the most frequent n-grams used in all sample works by that author with their normalized frequencies. So, a profile is a set of ordered pairs (x_i, f_i), where x_i is the *i*th most frequent n-gram used by that author and f_i is the normalized frequency of that n-gram. The number of n-grams in the set is L, so the set contains the L most frequently-used n-grams. Frantzeskou uses raw frequencies, rather than normalized frequencies, however. The contention is that the frequencies are not used except to sort the n-grams, so normalization is not necessary.

The n-grams are extracted at the byte level for programs in the data set, which means that all information stored in the source files are represented in the profiles – no information is lost. Whitespace, comments, every single byte saved in the source file is processed and included as n-grams in the author profiles.

The similarity measure used in the Frantzeskou method is the Simplified Profile Intersection (SPI). The SPI is simply a count of the number of n-grams that a profile and a query program have in common: $| P_A \cap P_p |$, where P_A represents the author profile and P_p represents the program profile. This simple metric is used to determine which author profile in the data set is most similar to a query program, and it is the author whose profile is most similar that is attributed to be the author. So, in essence, it is the author who frequently uses the sequences of characters that appear most frequently in the query program that is attributed to be the author.

2.2 The Burrows Method

The Burrows method uses a lossy approach for representing programs. In this method, n-grams are based on tokens. Tokens include selected operators, keywords, and white space. Programs are scanned (such that information deemed irrelevant is lost), and the token stream is broken into n-grams using a sliding window approach.

Based on empirical results, the authors of this method chose n=6 for the n-gram size. The similarity measure used is Okapi BM25 [15]. This measure was selected among five similarity measures that were evaluated: Okapi BM25, Cosine, Pivoted Cosine, language modeling with Dirichlet smoothing, and a metric developed specifically for source code authorship attribution called *Author1*. Through empirical testing, Okapi BM25 was found to be the most effective.

The actual approach for attributing authorship is typical for similarity-based authorship attribution methods. To determine the author of a program, that program is considered to be a query. The query is compared using a similarity measure to all of the programs in the data set. The author of the most-similar program is considered the author of the query program. So, in essence, it is the author who wrote the program that is most similar to the query program that is attributed to be the author.

3 The Comparative Study

Although both the Burrows and Frantzeskou methods have been shown to be state of the art, this is the first independent comparative study that has been performed on them. This study consisted of a 20-class experiment. A leave-one-out cross validation experimental design was used. The results were measured as a percentage of programs correctly identified. The data set included both C++ and Java programs.

The collection of programs used in the study included a total of 7517 Java and C++ documents. The programs consisted of sample programs distributed with introductory programming and data structures textbooks. The textbooks included twenty Java textbooks and twenty C++ textbooks. Among the Java textbooks, there were no duplicate authors. Similarly, among the C++ textbooks, there were no duplicate authors. There were, however, nine textbooks that overlapped between the two languages. That is, nine textbooks were selected that had a Java edition and an equivalent C++ edition. So, there were a total of 31 unique authors represented (11 unique to the Java collection, 11 unique to the C++ collection, and 9 that overlapped between the two languages). There were 3906 documents in the C++ collection and 3611 documents in the Java collection, meaning the C++ collection had an average of 195 documents per author while the Java collection had an average of 181 documents per author.

Sample programs from programming textbooks were used, in part, to provide an accessible analog to student-submitted programs. The programs are academic in nature, varied according to the nature of the material being exemplified in each sample program, and reasonably close to "perfect ground truth." Copyright laws and reputations would prohibit plagiarism. Consistency in approach and style would be self-enforced for reasons related to both pedagogy and software engineering. Moreover, sample programs from textbooks are generally feely available and easily accessible.

The study was conducted as a series of 20-class experiments, using a leave-one-out cross validation experimental design, and the results were measured as a percentage of programs correctly identified. A 20-class experiment means that the author of each document was determined from a set of 20 candidate authors. A leave-one-out cross validation experimental design means that each program in the data set was selected, in turn, as a query program while the remaining programs were used as training data. The results being measured in terms of accuracy means that the results were measured as a percentage of programs whose authors were correctly identified.

Each experiment was conducted as follows: Every program in the data set was represented as dictated by the method being evaluated. For the Burrows method, each program was tokenized. For the Frantezkou method, a source code author profile (SCAP) was created for each author. Each program in the data set was selected, in turn, as a query program. The author of that program was attributed according to the method being evaluated, using the remaining programs in the data set as the programs of known authorship. Each program was marked as either correctly attributed or incorrectly attributed. The overall results of the experiment were measured as a percentage of the programs correctly attributed.

The Frantzeskou method successfully attributed 94.3% of the documents, while the Burrows method successfully attributed 89.5% of the documents. One could argue, however, that this comparison is unfair. The Burrows method inherently anonymizes the data by removing all comments and string literals. By not anonymizing the data, the Frantzeskou method has an intrinsic advantage of including in its similarity calculations information contained therein. So, the methods were also compared using anonymized documents. When using anonymized documents, the overall results for the methods were quite similar.

In the end, a total of six individual experiments were conducted: (1) the Burrows method was used to attribute the C++ documents, (2) the Burrows method was used to attribute the Java documents, (3) the Frantzeskou method was used to attribute the C++ documents, (4) the Frantzeskou method was used to attribute the Java documents, (5) the Frantzeskou method was used to attribute anonymized versions of the C++ documents, and (6) the Frantzeskou method was used to attribute anonymized versions of the Java documents. The results of the study are shown in Figure 1.

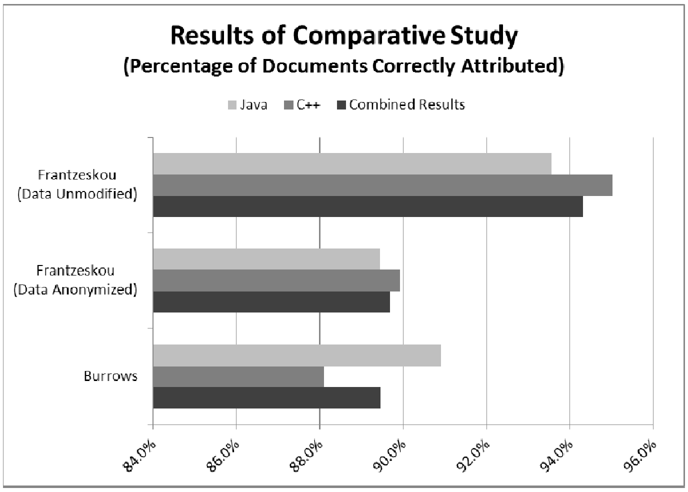

Fig. 1. Results of comparative study

Frantzeskou successfully attributed a larger percentage of the C++ documents, even after they were anonymized. However, Burrows attributed a larger percentage of the Java documents. Also, a relatively large discrepancy can be seen in the number of Java documents and C++ documents successfully attributed by Burrows. This discrepancy could be explained by the features selected for tokenization, because an utterly different set of features were used for each language. Feature selection is discussed further in the Analysis section.

4 Analysis

The two methods evaluated are clearly state of the art in authorship attribution of source code. In the worst case (the Burrows attribution of C++ documents), over 88% of the documents were successfully attributed. Given the large nature of the data set, these results are remarkable. However, opportunities to improve both methods clearly exist. This section describes some of those opportunities.

4.1 The Burrows Method

Perhaps the most obvious improvement to be made to the Burrows method is that of feature selection. The features were selected by creating six classes of features: operators, keywords, input/output tokens, function tokens, white space tokens, and literal tokens. Selected features were categorized into these classes. Sets of features were formed from all possible combinations of the classes, and empirical means were used to select the most significant feature classes. In the end, the feature classes selected were operators, keywords, and white space tokens.

An issue with this methodology is that the initial selection of features was somewhat arbitrary, as was the categorization into the six classes. Moreover, the fact that features were grouped and evaluated thusly meant that individual features were not evaluated – rather, somewhat arbitrary groupings of features were evaluated. In some of these classes, obvious omissions were made. For example, the white space tokens included carriage returns and new lines, but did not include line feeds. Furthermore, commonly-used symbols that are often emphasized in regards to programming style – such as semicolons and "curly braces" – were not even considered.

The Burrows method uses the "single best result" metric to assign authorship. That is, the author of the top-ranked document returned by the search query is attributed to be the author of said query document. This metric was selected based on an empirical comparison over two other metrics that were also considered. Additional metrics could certainly be considered. One possibility is to utilize an idea from the Frantzeskou method, and represent an author's entire corpus as a profile. If the work of each author were represented as a single document, it would certainly make sense for the author of the "single best result" to be attributed as the author.

In the Burrows experiments presented in this paper, the query documents were left in the corpus when the indexes were created by the search engine. As a result, the query document affects the Okapi similarity calculations. So, even though the query document was omitted from the results returned by the search engine, the query document still played a role in determining which results were returned in the first place. Therefore, the results of the comparative study are thusly skewed in favor of the Burrows method. For a better comparison, the query document itself should provide absolutely no knowledge in determining the authorship of said document.

4.2 The Frantzeskou Method

One potential improvement to be made to the Frantzeskou method is that of the similarity metric, the so-called SPI. The metric is used to determine which author profile in the data set is most similar to a query program, simply by determining how many n-grams the author profile and query program have in common. So, essentially, it is the author who frequently uses the sequences of characters that appear most frequently in the query program that is attributed to be the author. This similarity metric is quite simplistic, and a more sophisticated metric might be apropos.

In the Frantzeskou method, an author profile includes the *L* most frequently occurring n-grams used by that author, where *L* is a parameter. Choosing the size of *L* is a difficult task. Indeed, Frantzeskou leaves the determination of the optimal value for *L* to future work. Burrows suggests that the optimal value of *L* is effectively infinity, such that author profiles are not truncated at all, noting that this technique is equivalent to coordinate matching [16-17].

5 Conclusion and Future Work

The two methods evaluated are clearly state of the art in authorship attribution of source code. We've shown that the Frantzeskou method can successfully attribute over 94% of documents in a 20-class experiment. When the data has been anonymized by stripping out all comments and string literals, the success rate still approaches an impressive 90%. Likewise, the Burrows method, which inherently anonymizes data, successfully attributed nearly 90% of all documents.

Incremental improvements can likely be made to both methods. Selecting different feature sets, tweaking Okapi parameters, selecting an altogether different similarity measurement, utilizing a metric other than "single best result" to assign authorship, and representing an author's entire set of work as a profile rather than as individual documents would all be viable opportunities for improving the Burrows method. Potential improvements to the Frantzeskou method might include utilizing other similarity metrics and investigating the choice for the *L* parameter. When performing studies utilizing the Burrows method, it is also imperative that the query document not be included when generating the search engine indexes.

Future work also includes investigating ways of combining the two current state-of-the-art methods to create a new, more-effective method. Perhaps a "confidence level" could be applied to each document attribution. If the confidence is deemed low, alternative factors and/or methods could be utilized to assist in the final determination. Perhaps "identifying" data such as comments and string literals could be handled separately from the primary method of authorship attribution. If such information is unavailable, it won't affect the primary means of attribution. If it is available, it could be used in a secondary manner to supplement the primary method. This could make sense, considering the data contained in comments and string literals are mostly free from the confines of the programming language syntax and so, perhaps, should intrinsically be analyzed differently. Other factors such as identifiers and file names could potentially be analyzed in a similar way.

References

1. Spafford, E.H., Weeber, S.A.: Software Forensics: Can We Track Code to its Authors? Computers & Security (COMPSEC) 12(6), 585–595 (1993)
2. Zhao, Y., Zobel, J.: Effective and Scalable Authorship Attribution Using Function Words. In: Lee, G.G., Yamada, A., Meng, H., Myaeng, S.-H. (eds.) AIRS 2005. LNCS, vol. 3689, pp. 174–189. Springer, Heidelberg (2005)
3. Juola, P.: Authorship attribution. Foundations and Trends in Information Retrieval 1(3), 233–334 (2007)
4. Stamatatos, E.: A Survey of Modern Authorship Attribution Methods. Journal of the American Society for Information Science and Technology 60(3), 538–556 (2009)
5. Oman, P.W., Cook, C.R.: Programming Style Authorship Analysis. In: Proceedings of the 17th Conference on ACM Annual Computer Science Conference (CSC), pp. 320–326 (1989)
6. Gray, A., Sallis, P., MacDonell, S.: IDENTIFIED (Integrated Dictionary-based Extraction of Non-language Dependent Token Information for Forensic Identification, Examination, and Discrimination): A Dictionary-based System for Extracting Source Code Metrics for Software Forensics. In: Proceedings of the International Conference on Software Engineering (ICSE), pp. 252–259 (1998)
7. Krsul, I., Spafford, E.H.: Authorship Analysis: Identifying the Author of a Program. Computers & Security (COMPSEC) 16(3), 233–257 (1997)
8. MacDonell, S.G., Gray, A.R., MacLennan, G.,Sallis, P.J.: Software Forensics for Discriminating between Program Authors using Case-based Reasoning, Feedforward Neural Networks and Multiple Discriminant Analysis. In: Proceedings of the 6th International Conference on Neural Information Processing (ICONIP), pp. 66-71 (1999)
9. Ding, H., Samadzadeh, M.H.: Extraction of Java Program Fingerprints for Software Authorship Identification. The Journal of Systems and Software 72, 49–57 (2004)
10. Lange, R., Mancoridis, S.: Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 2082–2089 (2007)
11. Elenbogen, B.S., Seliya, N.: Detecting Outsourced Student Programming Assignments. Journal of Computing Sciences in Colleges 23(3), 50–57 (2008)
12. Frantzeskou, G., Stamatatos, E., Gritzalis, S., Chaski, C.E., Howald, B.S.: Identifying Authorship by Byte-Level N-Grams: The Source Code Author Profile (SCAP) Method. International Journal of Digital Evidence 6(1), 1–18 (2007)
13. Burrows, S.D.: Source Code Authorship Attribution. Dissertation. RMIT University, Melbourne, Australia (2010)
14. Keselj, V., Peng, F., Cercone, N., Thomas, C.: N-gram Based Author Profiles for Authorship Attribution. In: Proceedings of the Pacific Association for Computational Linguistics, pp. 255–264 (2003)
15. Robertson, S.E., Walker, S.: Okapi/Keenbow at TREC-8. In: Voorhees, E., Harman, D. (eds.) Proceedings of the Eighth Text Retrieval Conference, pp. 151–162. National Institute of Standards and Technology, Gaithersburg (1999)
16. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann, San Francisco (1999)
17. Uitdenbogerd, A.L., Zobel, J.: Music ranking techniques evaluated. In: Oudshoorn, M., Pose, R. (eds.) Proceedings of the Twenty-Fifth Australasian Computer Science Conference, pp. 275–283. Australian Computer Society, Melbourne (2002)