Source Code Author Identification Based on N-gram Author Profiles

Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis,
Sokratis Katsikas
Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering
University of the Aegean, Karlovasi, Samos, 83200, Greece
{gfran, stamatatos, sgritz, ska}@aegean.gr

Abstract. Source code author identification deals with the task of identifying the most likely author of a computer program, given a set of predefined author candidates. This is usually based on the analysis of other program samples of undisputed authorship by the same programmer. There are several cases where the application of such a method could be of a major benefit, such as authorship disputes, proof of authorship in court, tracing the source of code left in the system after a cyber attack, etc. We present a new approach, called the SCAP (Source Code Author Profiles) approach, based on byte-level n-gram profiles in order to represent a source code author's style. Experiments on data sets of different programming language (Java or C++) and varying difficulty (6 to 30 candidate authors) demonstrate the effectiveness of the proposed approach. A comparison with a previous source code authorship identification study based on more complicated information shows that the SCAP approach is language independent and that n-gram author profiles are better able to capture the idiosyncrasies of the source code authors. Moreover the SCAP approach is able to deal surprisingly well with cases where only a limited amount of very short programs per programmer is available for training. It is also demonstrated that the effectiveness of the proposed model is not affected by the absence of comments in the source code, a condition usually met in cyber-crime cases.

1 Introduction

Nowadays, in a wide variety of cases it is important to identify the author of a (usually limited) piece of code. Such situations include authorship disputes, proof of authorship in court, cyber attacks in the form of viruses, trojan horses, logic bombs, fraud, and credit card cloning etc. Although source code is much more formal and restrictive than spoken or written languages, there is still a large degree of flexibility

Please use the following format when citing this chapter:
Frantzeskou, Georgia, Stamatatos, Efstathios, Gritzalis, Stefanos, Katsikas, Sokratis, 2006, in IFIP International Federation for Information Processing, Volume204, Artificial Intelligence Applications and Innovations, eds. Maglogiannis, I., Karpouzis, K., Bramer, M., (Boston: Springer), pp. 508–515

when writing a program [6]. Source code author identification is much harder than natural language authorship attribution or writer identification (of handwriting) or even speaker recognition. The traditional methodology that has been followed in this area of research is divided into two main steps ([5, 7, 1]). The first step is the extraction of software metrics representing the author's style and the second step is using these metrics to develop models that are capable of discriminating between several authors, using a classification algorithm.

However, there are some disadvantages in this traditional approach. The first is that software metrics used are programming language dependant. For example metrics used in Java cannot be used in C or Pascal. The second is that metrics selection is not a trivial process and usually involves setting thresholds to eliminate those metrics that contribute little to the classification model. As a result, the focus in a lot of the previous research efforts, such as [1] and [5] was into the metrics selection process rather than into improving the effectiveness and the efficiency of the proposed models.

In this paper we present an approach to source code author identification we call the SCAP (Source Code Author Profiles) approach, which is an extension of a method that has been applied to natural language text authorship identification [3]. In the SCAP method, byte-level n-grams are utilised together with author profiles. We propose a new simplified profile and a less complicated similarity measure that proved to be quite effective even in cases where only limited training set is available for each author. Our methodology is programming language independent since it is based on low-level information and has been tested to data sets from two different programming languages Java and C++. Special attention is paid to the evaluation methodology. Disjoint training and test sets of equal size were used in all the experiments in order to ensure the reliability of the presented results. Moreover, the significance of the comments in the source code is examined. It is demonstrated that the effectiveness of the SCAP model is not affected by the absence of comments, a condition usually met in cyber-crime cases.

The rest of this paper is organized as follows. Section 2 describes our approach and section 3 includes the source code author identification experiments. Finally, section 4 contains conclusions and future work.

2 The SCAP Approach

In this paper, we present the SCAP (Source Code Author Profiles) approach, which is an extension of a method that has been successfully applied to text authorship identification [3]. It is based on byte level n-grams and the utilization of a similarity measure used to classify a program to an author. Therefore, this method does not use any language-dependent information.

An n-gram is an n-contiguous sequence and can be defined on the byte, character, or word level. Byte, character and word n-grams have been used in a variety of applications such as text authorship attribution, speech recognition, language modelling, context sensitive spelling correction, optical character recognition etc. In our approach, the Perl package Text::N-grams [4] has been used

to produce n-gram tables for each file or set of files that is required. The n-gram table contains the n-grams found in a source code file and their corresponding frequency of occurrence.

The algorithm used, computes n-gram based profiles that represent each of the author category. First, for each author the available training source code samples are concatenated to form a big file. Then, the set of the L most frequent n-grams of this file is extracted. The profile of an author is, then, the ordered set of pairs $\{(x_1; f_1), (x_2; f_2),\ldots,(x_L; f_L)\}$ of the L most frequent n-grams x_i and their normalized frequencies f_i. Similarly, a profile is constructed for each test case (a simple source code file). In order to classify a test case in to an author, the profile of the test file is compared with the profiles of all the candidate authors based on a similarity measure. The most likely author corresponds to the least dissimilar profile (in essence, a nearest-neighbour classification model).

The original similarity measure (i.e. dissimilarity more precisely) used by Keselj [3] in text authorship attribution is a form of relative distance:

$$\sum_{n\in profile}\left(\frac{f1(n)-f2(n)}{\frac{f1(n)+f2(n)}{2}}\right)^2=\sum_{n\in profile}\left(\frac{2(f1(n)-f2(n))}{f1(n)+f2(n)}\right)^2 \quad (1)$$

where $f_1(n)$ and $f_2(n)$ are the normalized frequencies of an n-gram n in the author and the program profile, respectively, or 0 if the n-gram does not exist in the profile. A program is classified to the author, whose profile has the minimal distance from the program profile, using this measure. Hereafter, this distance measure will be called Relative Distance (RD).

One of the inherent advantages of this approach is that it is language independent since it is based on low-level information. As a result, it can be applied with no additional cost to data sets where programs are written in C++, Java, perl etc. Moreover, it does not require multiple training examples from each author, since it is based on one profile per author. The more source code programs available for each author, the more reliable the author profile. On the other hand, this similarity measure is not suitable for cases where only a limited training set is available for each author. In that case, for low values of n, the possible profile length for some authors is also limited, and as a consequence, these authors have an advantage over the others. Note that this is especially the case in many source code author identification problems, where only a few short source code samples are available for each author.

In order to handle this situation, we introduce the SCAP approach. It includes a new similarity measure that does not use the normalized frequencies f_i of the n-grams. Hence the profile we propose is a Simplified Profile (SP) and is the set of the L most frequent n-grams $\{x_1, x_2,\ldots, x_L\}$. If SP_A and SP_P are the Author and Program Simplified Profiles, respectively, then the similarity distance is given by the size of the intersection of the two profiles:

$$\left|SP_A \cap SP_P\right| \quad (2)$$

where |X| is the size of X. In other words, the similarity measure we propose is the amount of common n-grams in the profiles of the test case and the author. The program is classified to the author with whom we achieved the biggest size of

intersection. Hereafter, this similarity measure will be called Simplified Profile Intersection (SPI). We have developed a number of perl scripts in order to create the sets of n-gram tables for the different values of n (i.e., n-gram length), L (i.e., profile length) and for the classification of the program file to the author with the smallest distance.

Table 1. The data sets used in this study. Program sample length is expressed by means of Lines Of Code (LOC)

	MacDonellC++	OSJava1	NoComJava	OnlyComJava	OSJava2
No Authors	6	8	8	6	30
Min-Max Samples per Author	5-114	4-29	4-29	9-25	4-29
Total Samples	268	107	107	92	333
Training Set Samples	134	56	56	46	170
Testing Set Samples	133	51	51	43	163
Size of smallest sample (LOC)	19	23	10	6	20
Size of biggest sample (LOC)	1449	760	639	332	980
Mean LOC in Training Set	206.4	155.48	122.28	64.58	170.84
Mean LOC in Test Set	213	134.17	95.92	56.48	173.03
Mean LOC/sample	210	145	109.1	60.53	172

3 Experiments

3.1 Comparison with a previous method

Our purpose during this phase was to check that the presented approach works at least equally well as the previous methodologies for source code author identification. For this reason, we run this experiment with a data set that has been initially used by Mac Donell [7] for evaluating a system for automatic discrimination of source code author based on more complicated, programming language-dependent measures. All the source code samples were written in C++. The source code for programmers one, two, and three were from programming books and programmers four, five, and six were experienced commercial programmers. Detailed information for the C++ data set is given in Table 1. The best reported result by Mac Donell [7] on the test set was 88% using the case-based reasoning (that is, a memory-based

learning) algorithm. Table 2 includes the classification accuracy results for various combinations of n (n-gram size) and L (profile size). In most cases, classification accuracy reaches 100%, much better than the best reported ([7]) accuracy for this data set (88% on the test set). This proves that the presented methodology can cope effectively with the source code author identification problem based on low-level information.

More importantly, RD performs much worse than SPI in all cases where at least one author profile is shorter than L. This occurs because the RD similarity measure (1) is affected by the size of the author profile. When the size of an author profile is lower than L, some programs are wrongly classified to that author. In summary, we can conclude that the RD similarity measure is not as accurate for those n, L combinations where L exceeds the size of even one author profile in the dataset. In all cases, the accuracy using the SPI similarity measure is better than (or equal to) that of RD. This indicates that this new and simpler similarity measure included in SCAP approach is not affected by cases where L is greater than the smaller author profile.

Table 2. Classification accuracy (%) on the MacDonellC++ data set for different values of n-gram size and profile size using RD and SPI similarity measures

Pro	n-gram Size											
file	3		4		5		6		7		8	
Size	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
1000	100	100	100	100	100	100	100	100	100	100	99	99
1500	100	100	100	100	100	100	100	100	99	99	99	100
2000	98	100	100	100	100	100	100	100	100	100	100	100
2500	99	100	100	100	100	100	100	100	100	100	100	100
3000	56	100	100	100	100	100	100	100	100	100	100	100

Table 3. Classification accuracy (%) on the OSJava1 data set

Pro	n-gram Size											
file	3		4		5		6		7		8	
Size	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
1500	88	100	100	100	100	100	100	100	100	100	100	100
2000	35	100	80	100	100	100	100	100	100	100	100	100

Table 4. Classification accuracy (%) on the NoComJava set

Pro	n-gram Size											
file	3		4		5		6		7		8	
Size	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
500	94	94	94	94	94	94	94	94	92	94	92	92
1500	35	98	47	90	80	98	96	98	98	98	98	98
2000	33	92	14	98	20	100	31	100	61	100	78	100

Table 5. C Classification accuracy (%) on the OnlyComJava data set

Pro	n-gram Size											
file	3		4		5		6		7		8	
Size	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
1500	98	98	98	98	98	100	95	95	95	95	95	98
2000	23	91	98	100	98	100	95	100	95	98	95	98

3.2 The role of comments

The experiments described in this section are based on a data set of open source programs written in Java. In more detail, source code samples by 8 different authors were downloaded from freshmeat.net. The amount of programs per programmer is highly unbalanced, ranging from 4 to 30 programs per author. The source code sample size was between 23-760 lines of code. In many cases, source code samples by the same programmer have common comment lines at the beginning of the program. Such comment lines were manually removed since they could (positively) influence the classification accuracy. The total number of programs was 107 and they were split into equally-sized training and test sets. Hereafter, this data set will be called OS-Java1.

This data set provides a more realistic case of source code author identification than student programs that have been used in similar studies ([2], [5]). Open source code is similar to commercial programs which usually have comments and they are usually well structured. Most of the open source programs are longer than the student programs. More importantly, this data set enables us to examine the role comments play in the classification model. We have decided to perform three different experiments on this data set. For this reason, we first filtered out any comments from the OSJava1 data set, resulting a new data set (hereafter, called NoComJava). Then, another data set was constructed using only the comments from each source code sample (hereafter, called OnlyComJava). Note that in the latter case, the resulting data set includes fewer programs than the original because any source code files with no comments were removed. The OnlyComJava data set includes samples by 6 different authors with 9 – 25 files per author. Detailed information for OSJava1, NoComJava, and OnlyComJava data sets is shown in Table 1.

The application of the proposed methodology to the OSJava1, NoComJava, and OnlyComJava data sets is described in Tables 3, 4, and 5, respectively. Notice that two different profile sizes are indicated (1500 and 2000) since they provide the best results (as has been demonstrated in previous study [2]). The classification results for the OSJava1 data set are perfect for any n-gram size and similarity measure. This is mainly because the source code samples of this data set are relatively long. Moreover, for many candidate authors there is a sufficient amount of training samples. Interestingly, the accuracy remains at the top level when removing the comment lines of these samples (NoComJava data set). However, this stands only for the SPI similarity measure. RD fails to retain such performance in most cases. In more detail, for L=500, RD and SPI have (almost) identical performance. When L increases to 1500, the accuracy of RD drops for low values of n (n<6). When L

increases to 2000, the accuracy of RD drops for all values of n. This happens because at least one author has author profile shorter than the predefined value of L. RD is not able to handle effectively such cases. Note that the accuracy of SPI increases with L. This is a strong indication that the proposed SPI similarity measure better suits the source code author identification problem. On the other hand, when examining only the comments of each source code sample (OnlyComJava dataset), the RD similarity measure is more competitive, which indicates that it better suits natural language.

Table 6. Classification accuracy (%) on the OSJava2 data set

Pro	n-gram Size											
file	3		4		5		6		7		8	
Size	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
1000	93	93	93	94	95	95	94	94	96	95	94	94
1500	92	92	94	94	95	95	96	96	96	97	95	95
2000	31	92	72	94	95	95	95	94	95	96	96	96
2500	13	93	37	94	54	95	79	94	94	94	95	95
3000	14	89	13	94	24	95	38	95	58	95	75	95

Again, the best results are obtained using the SPI measure. Probably, this is explained by the extremely short samples that constitute the OnlyComJava data set.

3.3 Dealing with many authors

The previous experiments have shown that our approach is quite reliable is quite reliable when dealing with a limited number of candidate authors (6 to 8). In this section we present an experiment that demonstrates the effectiveness of the proposed method when dealing with dozens of candidate authors. For that purpose a data set was created by downloading open-source code samples by 30 different authors from freshmeat.net. Hereafter, this data set will be called OSJava2. Details on this data set can be found in Table 1. Note that the available texts per author ranges from 4 to 29. Moreover, in average the samples of this data set are longer in comparison to the OSJava1. This data set includes programs on the same application domain written by different authors. In addition the samples of many authors are written over a long time period and therefore there might be programming style changes of certain authors.

The samples were split into equally-sized training and test set. Note that the training set was highly unbalanced (as OSJava1). The best accuracy result was 96.9% and has been achieved using the SPI similarity measure as can be seen in Table 6. Again, RD fails to deal with cases where at least one author profile is shorter than L. In most cases, accuracy exceeds 95%, using the SPI similarity measure indicating that the SCAP approach can reliably identify the author of a source code sample even when there are multiple candidate authors. The best result corresponds to profile size of 1500.

4 Conclusions

In this paper, the SCAP approach to source code authorship analysis has been presented. It is based on byte-level n-gram profiles, a technique successfully applied to natural language author identification problems. This method was applied to data sets of varying difficulty demonstrating surprising effectiveness. The SCAP approach includes a new simplified profile and a less-complicated similarity measure that better suit the characteristics of the source code authorship analysis problem. In particular the SCAP approach can deal with cases where very limited training data per author is available (especially, when at least one author profile is shorter than the predefined profile size) or there are multiple candidate authors, conditions usually met in source code authorship analysis problems (e.g. source code authorship disputes, etc.) with no significant compromise in performance.

More significantly, the role of comments in the source code is examined. The SCAP method can reliably identify the most likely author when there are no comments in the available source code samples, a condition usually met in cyber-attacks. However, it is demonstrated that the comments provide quite useful information and can significantly assist the classification model to achieve quasi-perfect results. Actually, the comments alone can be used to identify the most likely author in open-source code samples where there are detailed comments in each program sample.

A useful direction for further work would be the discrimination of different programming styles in collaborative projects. In addition, cases where all the available source code programs are dealing with the same task should be tested. Finally, the visualization of the stylistic properties of each author could be of major benefit in order to explain the differences between candidate source code authors.

References

1. B Ding, H., Samadzadeh, M., H., Extraction of Java program fingerprints for software authorship identification, The Journal of Systems and Software, Volume 72, Issue 1, Pages 49-57 June 2004.
2. Frantzeskou, G., Stamatatos, E., Gritzalis, S., Supporting the cybercrime investigation process: Effective discrimination of source code based on byte level information, in Proc. 2nd International Conference on e-business and Telecommunications Networks (ICETE05), 2005.
3. Keselj, V., Peng, F., Cercone, N., Thomas, C., N-gram based author profiles for authorship attribution, In Proc. Pacific Association for Computational Linguistics 2003.
4. Keselj, V.,. Perl package Text::N-grams http://www.cs.dal.ca/~vlado/srcperl/N-grams or http://search.cpan.org/author/VLADO/Text-N-grams-0.03/N-grams.pm, 2003.
5. Krsul, I., and Spafford, E. H, Authorship analysis: Identifying the author of a program, In Proc. 8th National Information Systems Security Conference, pages 514-524, National Institute of Standards and Technology, 1995.
6. Krsul, I., and Spafford, E. H., 1996, Authorship analysis: Identifying the author of a program, Technical Report TR-96-052, 1996.
7. MacDonell, S.G, and Gray, A.R. Software forensics applied to the task of discriminating between program authors. Journal of Systems Research and Information Systems 10: 113-127 (2001).