SUPPORTING THE CYBERCRIME INVESTIGATION PROCESS: EFFECTIVE DISCRIMINATION OF SOURCE CODE AUTHORS BASED ON BYTE-LEVEL INFORMATION

Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis
Laboratory of Information and Communication Systems Security, Aegean University
Department of Information and Communication Systems Engineering, Karlovasi, Samos, 83200, Greece

Keywords: Source Code Authorship Analysis, Software Forensics, Security.

Abstract: Source code authorship analysis is the particular field that attempts to identify the author of a computer program by treating each program as a linguistically analyzable entity. This is usually based on other undisputed program samples from the same author. There are several cases where the application of such a method could be of a major benefit, such as tracing the source of code left in the system after a cyber attack, authorship disputes, proof of authorship in court, etc. In this paper, we present our approach which is based on byte-level n-gram profiles and is an extension of a method that has been successfully applied to natural language text authorship attribution. We propose a simplified profile and a new similarity measure which is less complicated than the algorithm followed in text authorship attribution and it seems more suitable for source code identification since is better able to deal with very small training sets. Experiments were performed on two different data sets, one with programs written in C++ and the second with programs written in Java. Unlike the traditional language-dependent metrics used by previous studies, our approach can be applied to any programming language with no additional cost. The presented accuracy rates are much better than the best reported results for the same data sets.

1 INTRODUCTION

In a wide variety of cases it is important to identify the author of a piece of code. Such situations include cyber attacks in the form of viruses, trojan horses, logic bombs, fraud, and credit card cloning or authorship disputes or proof of authorship in court etc. But why do we believe it is possible to identify the author of a computer program? Humans are creatures of habit and habits tend to persist. That is why, for example, we have a handwriting style that is consistent during periods of our life, although the style may vary, as we grow older. Does the same apply to programming? Although source code is much more formal and restrictive than spoken or written languages, there is still a large degree of flexibility when writing a program (Krsul, and Spafford, 1996).

Source code authorship analysis could be applied to the following application areas (Frantzeskou et al 2004):

1. *Author identification*. The aim here is to decide whether some piece of code was written by a certain author. This goal is accomplished by comparing this piece of code against other program samples written by that author. This type of application area has a lot of similarities with the corresponding literature where the task is to determine that a piece of work has been written by a certain author.

2. *Author characterisation*. This application area determines some characteristics of the author of a piece of code, such as cultural educational background and language familiarity, based on their programming style.

3. *Plagiarism detection*. This method attempts to find similarities among multiple sets of source code files. It is used to detect plagiarism, which can be defined as the use of another person's work without proper acknowledgement.

4. *Author discrimination*. This task is the opposite of the above and involves deciding whether some pieces of code were written by a single author

Frantzeskou G., Stamatatos E. and Gritzalis S. (2005).
SUPPÎŞRTING THE CYBERCRIME INVESTIGATION PROCESS: EFFECTIVE DISCRIMINATION OF SOURCE CODE AUTHORS BASED ON BYTE-LEVEL INFORMATION.
In *Proceedings of the Second International Conference on e-Business and Telecommunication Networks*, pages 283-290
DOI: 10.5220/0001414902830290

or by some number of authors. An example of this would be showing that a program was probably written by three different authors, without actually identifying the authors in question.

5. Author intent determination. In some cases we need to know whether a piece of code, which caused a malfunction, was written having this as its goal or was the result of an accidental error. In many cases, an error during the software development process can cause serious problems.

The traditional methodology that has been followed in this area of research is divided into two main steps (Krsul, Spafford 1995; MacDonell et al. 2001; Ding 2004). The first step is the extraction of software metrics and the second step is using these metrics to develop models that are capable of discriminating between several authors, using a machine learning algorithm. In general, the software metrics used are programming - language dependent. Moreover, the metrics selection process is a non trivial task.

In this paper we present a new approach, which is an extension of a method that has been applied to natural language text authorship identification (Keselj et al., 2003). In our method, byte-level N-grams are utilised together with author profiles. We propose a new simplified profile and a new similarity measure which enables us to achieve a high degree of accuracy for authors for whom we have a very small training set. Our methodology is programming - language independent since it is based on low-level information and is tested to data sets from two different programming languages. The simplified profile and the new similarity measure we introduce provide a less complicated algorithm than the method used in text authorship attribution and in many cases they achieve higher prediction accuracy. 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. Note, that in many previous studies the evaluation of the proposed methodologies was performed on the training set. Our approach is able to deal effectively with cases where there are just a few available programs per author. Moreover, the accuracy results are high even for cases where the available programs are of restricted length.

The rest of this paper is organized as follows. Section 2 contains a review on past research efforts in the area of source code authorship analysis. Section 3 describes our approach and section 4 includes the experiments we have performed. Finally, section 5 contains conclusions and future work.

2 RELATED WORK

The most extensive and comprehensive application of authorship analysis is in literature. One famous authorship analysis study is related to Shakespeare's works and is dating back over several centuries. Elliot and Valenza (1991) compared the poems of Shakespeare and those of Edward de Vere, 7th Earl of Oxford, where attempts were made to show that Shakespeare was a hoax and that the real author was Edward de Vere, the Earl of Oxford. Recently, a number of authorship attribution approaches have been presented (Stamatatos et. al, 2000; Keselj, et al., 2003; Peng et al, 2004) proving that the author of a natural language text can be reliably identified.

Although source code is much more formal and restrictive than spoken or written languages, there is still a large degree of flexibility when writing a program (Krsul, and Spafford, 1996). Spafford and Weeber (1993) suggested that it might be feasible to analyze the remnants of software after a computer attack, such as viruses, worms or trojan horses, and identify its author. This technique, called software forensics, could be used to examine software in any form to obtain evidence about the factors involved. They investigated two different cases where code remnants might be analyzed: executable code and source code. Executable code, even if optimized, still contains many features that may be considered in the analysis such as data structures and algorithms, compiler and system information, programming skill and system knowledge, choice of system calls, errors, etc. Source code features include programming language, use of language features, comment style, variable names, spelling and grammar, etc.

Oman and Cook (1989) used "markers" based on typographic characteristics to test authorship on Pascal programs. The experiment was performed on 18 programs written by six authors. Each program was an implementation of a simple algorithm and it was obtained from computer science textbooks. They claimed that the results were surprisingly accurate.

Longstaff and Shultz (1993) studied the WANK and OILZ worms which in 1989 attacked NASA and DOE systems. They have manually analyzed code structures and features and have reached a

conclusion that three distinct authors worked on the worms. In addition, they were able to infer certain characteristics of the authors, such as their educational backgrounds and programming levels. Sallis et al (1996) expanded the work of Spafford and Weeber by suggesting some additional features, such as cyclomatic complexity of the control flow and the use of layout conventions.

An automated approach was taken by Krsul and Spafford (1995) to identify the author of a program written in C. The study relied on the use of software metrics, collected from a variety of sources. They were divided into three categories: layout, style and structure metrics. These features were extracted using a software analyzer program from 88 programs belonging to 29 authors. A tool was developed to visualize the metrics collected and help select those metrics that exhibited little within-author variation, but large between-author variation. A statistical approach called discriminant analysis (SAS) was applied on the chosen subset of metrics to classify the programs by author. The experiment achieved 73% overall accuracy.

Other research groups have examined the authorship of computer programs written in C++ (Kilgour et al., 1997); (MacDonell et al. 2001), a dictionary based system called IDENTIFIED (integrated dictionary- based extraction of non-language-dependent token information for forensic identification, examination, and discrimination) was developed to extract source code metrics for authorship analysis (Gray et al., 1998). Satisfactory results were obtained for C++ programs using case-based reasoning, feed-forward neural network, and multiple discriminant analysis (MacDonell et al. 2001). The best prediction accuracy has been achieved by Case-Based Reasoning and it was 88% for 7 different authors.

Ding (2004), investigated the extraction of a set of software metrics of a given Java source code, that could be used as a fingerprint to identify the author of the Java code. The contributions of the selected metrics to authorship identification were measured by a statistical process, namely canonical discriminant analysis, using the statistical software package SAS. A set of 56 metrics of Java programs was proposed for authorship analysis. Forty-six groups of programs were diversely collected. Classification accuracies were 62.7% and 67.2% when the metrics were selected manually while those values were 62.6% and 66.6% when the metrics were chosen by SDA (stepwise discriminant analysis).

The main focus of the previous approaches was the definition of the most appropriate measures for representing the style of an author**.** Quantitative and qualitative measurements, referred to as metrics, are collected from a set of programs. Ideally, such metrics should have low within-author variability, and high between-author variability (Krsul and Spafford, 1996), (Kilgour et al., 1997). Such metrics include:

- Programming layout metrics: include those metrics that deal with the layout of the program. For example metrics that measure indentation, placement of comments, placement of braces etc.
- Programming style metrics: Such metrics include character preferences, construct preferences, statistical distribution of variable lengths and function name lengths etc.
- Programming structure metrics: include metrics that we hypothesize are dependent on the programming experience and ability of the author. For example such metrics include the statistical distribution of lines of code per function, ratio of keywords per lines of code etc.
- Fuzzy logic metrics: include variables that they allow the capture of concepts that authors can identify with, such deliberate versus non deliberate spelling errors, the degree to which code and comments match, and whether identifiers used are meaningful.

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 (Ding 2004) and (Krsul, Spafford 1995) was into the metrics selection process rather than into improving the effectiveness and the efficiency of the proposed models.

3 OUR APPROACH

In this paper, we present our approach, which is an extension of a method that has been successfully applied to text authorship identification (Keselj, et al 2003). It is based on byte level n-grams and the utilization of two different similarity measures 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 (Keselj 2003) has been used to produce n-gram tables for each file or set of files that is required. An example of such a table is given in Table 1. The first column contains the n-grams found in a source code file and the second column the corresponding frequency of occurrence.

Tab1e 1: n-gram frequencies extracted from a source code file.

3-gram	Frequency
sio	28
_th	28
f_(	20
_=_	17
usi	16
_ms	16
out	15
ine	15
\n/*	15
on_	14
_in	14
fp_	14
the	14
sg_	14
_i_	14
in_	14

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 et al (2003) 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 propose a new similarity measure that does not use the normalized differences 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 just 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.

4 EXPERIMENTS

4.1 Comparison with a Previous Approach

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 et al (2001) for evaluating a system for automatic discrimination of source code author based on more complicated, language-dependent measures. All programs were written in C++. The source code for the first three authors was taken from programming books while the last three authors were expert professional programmers. The data set was split (as equally as possible) into the training set 50% (134 programs) and the test set 50% (133 programs). The best result reported by Mac Donell et al (2001) on the test set was 88% using the case-based reasoning (that is, a memory-based learning) algorithm. Detailed information for the C++ data set is given in Table 2. Moreover, the distribution of the programs per author is given in Table 3.

Table 2. The data sets used in this study. 'Programs per author' is expressed by the minimum and maximum number of programs per author in the data set. Program length is expressed by means of Lines Of Code (LOC).

Data Set	**C++**	**Java**
Number of authors	6	8
Programs per author	5-114	5-8
Total number of programs	268	54
Training set programs	134	28
Testing set programs	133	26
Size of smallest program (LOC)	19	36
Size of biggest program (LOC)	1449	258
Mean LOC per program	210	129
Mean LOC in training set	206.4	131.7
Mean LOC in testing set	213	127.2

Table 3. Program distribution per author for the C++ data set.

	Training Set	**Test Set**
Author 1	34	34
Author 2	57	57
Author 3	13	13
Author 4	6	6
Author 5	3	2
Author 6	21	21

We used byte-level n-grams extracted from the programs in order to create the author and program profiles as well as the author and program simplified profiles. Table 4 includes the classification accuracy results for various combinations of n (n-gram size) and L (profile size). In many cases, classification accuracy reaches 100%, much better than the best reported (MacDonell et al, 2001) accuracy for this data set (88% on the test set). This proves that the presented methodology can cope with effectively with the source code author identification problem. For $n<4$ and $L<1000$ accuracy drops. The same (although to a lower extent) stands for $n>6$.

More importantly, RD performs much worse than SPI in all cases where at least one author profile is shorter than L. For example for L=1000 and n=2, L is greater than the size of the profile of Author No5 (the maximum L of the profile of Author No 5 is 769) and the accuracy rate declines to 51%. This occurs because the RD similarity measure (1) that calculates similarity 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 proves that this new and simpler similarity measure is not affected by cases where L is greater than the smaller author profile.

4.2 Application to a Different Programming Language

The next experiment was performed on a different data set from a different programming language. In more detail the new data set consists of student programs (assignments from a programming language course) written in Java. Detailed information for this data set is given in Table 2. We used 8 authors. From each author 6-8 programs were chosen. Table 5 shows the distribution of programs per author. The size of programs was between 36 and 258 lines of code. The data set was split in training and test set of approximately equal size. This data set has been chosen in order to evaluate our approach when the available training data per author are limited (6-7 short programs per author).

Table 4. Classification accuracy (%) on the C++ data set for different values of n-gram size and profile size using two similarity measures: Relative Distance and Simplified Profile Intersection.

Profile Size L	n-gram Size													
	2		3		4		5		6		7		8	
	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
200	98.4	98.4	97.7	97.7	97	97	95.5	95.5	94.7	95.5	92.5	92.5	92.5	94.7
500	100	100	100	100	100	100	99.2	100	98.4	98.4	97.7	97.7	97.7	97.7
1000	51	99.2	100	100	100	100	100	100	100	100	100	100	99.2	99.2
1500	5.3	98.4	100	100	100	100	100	100	100	100	99.2	99.2	99.2	100
2000	1.5	97.7	98.4	100	100	100	100	100	100	100	100	100	100	100
2500	1.5	95.5	99.2	100	100	100	100	100	100	100	100	100	100	100
3000	1.5	95.5	55.6	100	100	100	100	100	100	100	100	100	100	100

Note that the programs written by students usually have no comments, their programming style is influenced by the instructor, they can be plagiarised, circumstances that create some extra difficulties in the analysis

Table 5. Program distribution per author of the Java data set.

	Training Set	Test Set
Author 1	3	3
Author 2	4	4
Author 3	3	2
Author 4	3	3
Author 5	4	4
Author 6	3	3
Author 7	4	3
Author 8	4	4

The results of the proposed method to this data set are given in Table 6. The best accuracy rate achieved with similarity measure RD was 84.6%. Again, when the profile size of at least one author is shorter than the selected profile size L, the accuracy of RD drops significantly. Using the similarity measure SPI, the best result was 88.5%. In generally SPI performed better than RD. Moreover, it seems that $4<n<7$ and $1000<L<3000$ provide the best accuracy results.

4.3 The Significance of Training Set Size

The purpose of this experiment was to examine the degree in which the training set size affects the classification accuracy. For this reason we used the C++ data set for which we reached classification accuracy of 100% for many n, L combinations with both similarity measures. This result has been achieved by using a training set of 134 programs in total. For the purposes of this experiment we used the same test set as in the experiment of section 4.1 but now we used training sets of different, smaller size. The smallest training set was comprised by only one program from each author and the biggest by 5 programs from each one (with the exception of one author for whom the available training programs were only 3). The presented source code author identification approach was applied to these new training sets using n=6 and L=1500 and similarity measure SPI. Note that the training size of authors was smaller than L in many of these experiments and as already explained, in such cases the classification accuracy decreases dramatically when using the similarity measure RD.

The accuracy results achieved are shown in Table 7. As can be seen, even with just one program per author available in the training set, high classification accuracy was achieved. By adding a second program per author the accuracy increased significantly above 96%. Note that the second programs added in the training set were in average longer than the first programs (see second column in table 7). We reached 100% of accuracy for training set based on five programs per author. This is a strong indication that our approach is quite effective even when very limited size of training set is available; a condition usually met in source code author identification problems.

Table 6. Classification accuracy (%) on the Java data set for different values of n-gram size and profile size using two similarity measures: Relative Distance and Simplified Profile Intersection.

Profile Size L	n-gram Size											
	3		4		5		6		7		8	
	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI	RD	SPI
1000	80.8	80.8	84.6	84.6	84.6	84.6	80.8	80.8	80.8	80.8	84.6	84.6
1500	84.6	84.6	76.9	76.9	80.8	80.8	84.6	84.6	80.8	80.8	80.8	80.8
2000	53.8	80.8	65.4	80.8	76.9	80.8	84.6	88.5	84.6	84.6	84.6	84.6
2500	53.8	73.1	53.8	76.9	53,8	80.8	84.6	88.5	84.6	88.5	84.6	84.6
3000	53.8	73.1	53.8	80.8	50	76.9	53.8	84.6	69,2	84.6	84.6	84.6

Table 7. Classification Accuracy (%) on the C++ data set using different training set size (in programs per author).

Training Set Size	Mean LOC in Training Set	Accuracy (%)
1	52	63.9
2	212	96.2
3	171	97
4	170	99.2
5	197	100

5 CONCLUSIONS

In this paper, an 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. The accuracy achieved for two data sets from different programming languages were 88.5% and 100% on test sets disjoint from training set, improving the best reported results for this task so far. Moreover the proposed method is able to deal with very limited training data, a condition usually met in source code authorship analysis problems (e.g., cyber attacks, source code authorship disputes, etc.) with no significant compromise in performance.

We introduced a new simplified profile and a new similarity measure. The advantage of the new measure over the original similarity measure is that it is not dramatically affected in cases where there is extremely limited training data for some authors. Moreover, the proposed method is less complicated than the original approach followed in text authorship attribution.

More experiments have to be performed on various data sets in order to be able to define the most appropriate combination of n-gram size and profile size for a given problem. The role of comments has also to be examined. In addition, cases where all the available source code programs are dealing with the same task should be tested as well. Another useful direction would be the discrimination of different programming styles in collaborative projects.

REFERENCES

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,

Elliot, W., and. Valenza, R.,1991, *Was the Earl of Oxford The True Shakespeare?*, Notes and Queries, 38:501-506.

Gray, A., Sallis, P., and 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 SE:E&P'98 (Software Engineering: Education and Practice Conference)*, IEEE Computer Society Press, pages 252–259., 1998.

Gray, A., Sallis, P., and MacDonell, S., *Software forensics: Extending authorship analysis techniques to computer programs,* in Proc. 3rd Biannual Conf. Int. Assoc. of Forensic Linguists (IAFL'97), pages 1-8, 1997.

Frantzeskou, G., Gritzalis, S., Mac Donell, S., *Source Code Authorship Analysis for supporting the cybercrime investigation process*, in Proc. 1st International Conference on e-business and Telecommunications Networks (ICETE04), Vol 2, pages (85-92), 2004.

Keselj, V., Peng, F., Cercone, N., Thomas, C., *N-gram based author profiles for authorship attribution*, In Proc. Pacific Association for Computational Linguistics, 2003.

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.

Kilgour, R. I., Gray, A.R., Sallis, P. J., and MacDonell, S. G., *A Fuzzy Logic Approach to Computer Software Source Code Authorship Analysis,* In the Fourth International Conference *on* Neural Information Processing -- The Annual Conference of the Asian Pacific Neural Network Assembly (ICONIP'97). Dunedin. New Zealand, 1997.

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.

Krsul, I., and Spafford, E. H., 1996, *Authorship analysis: Identifying the author of a program*, Technical Report TR-96-052, 1996

Longstaff, T. A., and Schultz, E. E., *Beyond Preliminary Analysis of the WANK and OILZ Worms: A Case Study of Malicious Cod*e, Computers and Security, 12:61-77, 1993.

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)

Oman, P., and Cook, C.*, Programming style authorship analysis*. In Seventeenth Annual ACM Science Conference Proceedings, pages 320–326. ACM, 1989.

Peng, F., D., Shuurmans, and S., Wang., *Augmenting naive bayes classifiers with statistical language models,* Information Retrieval Journal, 7(1): 317-345, 2004.

Sallis P., Aakjaer, A., and MacDonell, S., *Software Forensics: Old Methods for a New Science*. Proceedings of SE:E&P'96 (Software Engineering: Education and Practice*)*. Dunedin, New Zealand, IEEE Computer Society Press, 367-371, 1996

Spafford, E. H., *The Internet Worm Program: An Analysis*," Computer Communications Review, 19(1): 17-49, 1989.

Spafford, E. H., and Weeber, S. A., *Software forensics: tracking code to its authors*, Computers and Security, 12:585-595, 1993

Stamatatos, E., N., Fakotakis, and G. Kokkinakis. *Automatic text categorisation in terms of genre and author*. Computational Linguistics, 26(4): 471-495, 2000.