Author Identification in Imbalanced Sets of Source Code Samples

Evangelos Chatzicharalampous, Georgia Frantzeskou, and Efstathios Stamatatos
Dept. of Information and Communication Systems Eng.
University of the Aegean
Karlovassi, Greece
email: e.chatz@hotmail.com; gfran@aegean.gr; stamatatos@aegean.gr

***Abstract*—Similarly to natural language texts, source code documents can be distinguished by their style. Source code author identification can be viewed as a text classification task given that samples of known authorship by a set of candidate authors are available. Although very promising results have been reported for this task, the evaluation of existing approaches avoids focusing on the class imbalance problem and its effect on the performance. In this paper, we present a systematic experimental study of author identification in skewed training sets where the training samples are unequally distributed over the candidate authors. Two representative author identification methods are examined, one follows the profile-based paradigm (where a single representation is produced for all the available training samples per author) and the other follows the instance-based paradigm (where each training sample has its own individual representation). We examine the effect of the source code representation on the performance of these methods and show that the profile-based method is better able to handle cases of highly skewed training sets while the instance-based method is a better choice in balanced or slightly-skewed training sets.**

Keywords: Source code author identification, class imbalance, byte-level n-grams

I. INTRODUCTION

Similarly to natural language texts, source code documents can be distinguished by their style [1]. Source code author identification can be seen as a text classification task given that samples of known authorship by a set of candidate authors are available [2]. Burrows [3] presents an excellent review of software forensics applications associated with this task. These include assisting the revealing of academic dishonesty cases, resolving disputes and litigation about source code samples, tracing the authors of malicious software, and assisting the maintenance of large software projects by assigning source code samples to contributors.

So far, existing approaches have focused on the definition of features to quantify the style of source code and effective classification models [4, 5, 6]. Some very promising results have been reported when dealing with short samples of code, multiple candidate authors and several programming languages, including C, C++, Java, and Lisp [2, 3, 7]. To evaluate the proposed models, the published studies use custom-built source code collections that include either *balanced training sets* (where the training samples are equally distributed over the candidate authors) or *imbalanced (skewed) training* sets (where some candidate authors are over-represented or under-represented in the training samples). Note also that even in case the samples are (almost) evenly distributed over the authors, the size of each sample may vary. Thus, long samples (multiple lines of code) may be available for some authors and short samples for other author. This can also be viewed as a case of class imbalance. Therefore, a more accurate definition of a skewed training set should also account for the lines of code (or KBs) of training samples per author. Existing studies avoid focusing on the class imbalance problem and its effect on the performance of source code author identification methods [3,4,7].

The software forensics applications associated with this task usually provide samples of known authorship that are unevenly distributed over the candidate authors. The degree of class imbalance (the ratio between the most over-represented author and the most under-represented author) may be anywhere from very low to very high while the number of over(under)-represented authors may also vary. Given the nature of these applications it is not easy (or even possible) to require additional code samples for some authors. Moreover, in forensic applications the availability of many source samples for some authors should not increase the probability of assigning a sample of unknown authorship to them (in other words, the priors should not be taken into account). It is, therefore, crucial to examine the performance of existing source code author identification methods under different scenarios of class imbalance so that to estimate their stability and extract conclusions about tuning their parameters according to the distribution of the training set.

In this paper we present a systematic experimental study of the effect of the class imbalance problem on source code author identification methods. A simple and programming language-independent code representation technique that is based on byte-level *n*-grams [2] is adopted and tested under a variety of scenarios. Two representative author identification methods are examined, one follows the *profile-based paradigm* (where a single representation is produced for all the available training samples per author) and the other follows the *instance-based paradigm* (where each training sample has its own individual representation) [8]. We examine the performance of these methods when the degree of class imbalance and the number of over(under)-represented authors vary and provide insight for choosing the most appropriate source code author identification method for a specific case with a certain training set distribution over the candidate authors.

The rest of this paper is organized as follows. The next Section discusses relevant work and Section 3 describes the source code author identification methods that will be studied.

Then, Section 4 presents the data and experiments performed with skewed training sets while Section 5 summarizes the conclusions drawn from this study and discusses future work directions.

II. Relevant Work

Source code author identification can be viewed as a typical text categorization task where a set of training samples belonging to a set of classes (authors) are given. Based on this training set and an appropriate quantification of source code samples a classification model is learned and then this is used to classify unknown samples to one of the candidate authors. Most of the published studies dealing with this task focus on the definition of appropriate metrics to quantify the style of source code samples.

To this end, one direction is to use software metrics that fall into three main categories: programming layout metrics (e.g., indentation measures, whitespace use etc.), programming style metrics (e.g., variable length measures, use of capital letters, etc.), and programming structure metrics (e.g., function length measures, use of complex branching constructs, etc) [1, 4, 5, 6]. Other more complicated measures involve subjective judgments of human-experts (e.g., degree of match between code and comments, use of meaningful identifier names, etc). Some of the software metrics are language-dependent, so they have to be modified in order to be applied to code in a certain programming language [5]. The studies based on such measures produce a low dimensional representation of code consisting of a few dozen of features.

A computationally simpler approach is the extraction of *n*-gram measures from source code. Byte-level *n*-grams (i.e., overlapping sequences of length *n*) have been proved to be very effective for representing the stylistic properties of code [2, 7]. They produce a high dimensional representation with a few thousand of features and they are language-independent. Another proposed method concerns the extraction of token *n*-grams from code, where each token may refer to an operator, a keyword, a function etc. [3].

In author identification literature there are two main paradigms of classification methods [8]. The profile-based methods attempt to capture the style of each author and handle all the training samples per author collectively. Thus, the differences between training samples of the same author are disregarded [2, 6]. On the other hand, the instance-based methods attempt to capture the style of each sample and handle each training sample separately. Even in cases there is only one training sample for a certain candidate author, these methods require it to be split into multiple sub-samples. The majority of published studies in source code author identification follow the latter approach. To this end, several classification algorithms have been used including discriminant analysis [1, 5], neural networks [4], nearest-neighbor using the OKAPI BM25 similarity measure [3].

The class imbalance problem arises when the training set is unequally distributed over the classes. To deal with this problem, there are two main approaches that attempt to balance the training set [9]: *down-sampling* (reducing the training samples of all classes so that to be equal to the most under-represented class) and *over-sampling* (increasing the training samples of the under-represented classes by repeating some of them). Since some classification methods disregard repetitions of the same instance, an alternative method produces additional synthetic data by adding small random values in some instances belonging to the initial training set [10]. In the framework of the author identification task, the training set can be enriched using synthetic data that can easily be available by segmenting texts according to class size or re-sampling training texts (i..e, using the same text lines multiple times in different training instances) [11].

Previous studies on source code author identification have only superficially dealt with the class imbalance problem. In some cases the datasets used for evaluating the existing approaches were (nearly-)balanced. Lange and Mancoridis [6] used a dataset consisting of 3 projects per author while Frantzeskou et al. used a collection of 5-8 samples per author [2, 7]. In other cases, the datasets are highly imbalanced. MacDonell and Gray [4] used a highly imbalanced dataset comprising 5-114 samples per author. They split this dataset in stratified training and test sets, thus favoring the identification of authors with many training samples. To avoid this problem Burrows [3] used the leave-one-out cross-validation methodology on three datasets comprising 14-26, 5-30, and 3-26 samples per author. Although this approach is certainly fairer and provides a more robust look on the effectiveness of the identification method, it fails to focus on the properties of the skewed dataset. Hence, it is not possible to extract any conclusions about the stability of the identification method when the distribution of the training samples over the author changes.

III. Source Code Author Identification

A. Profile-based Paradigm

The profile-based paradigm attempts to capture the style of authors rather than individual samples [8]. It disregards the differences between training samples by the same author and produces one single representation (i.e., profile) per author. Each text of unknown authorship is then compared with the profile of each author and is assigned to the most likely one according to a similarity function. This procedure is illustrated in Figure 1. The profile-based methods are affected by the class imbalance problem when there is an uneven distribution of the total size (in KBs) of training samples over the authors. On the other hand, the number of training samples per author is not significant. Since they concatenate all the available training samples per author, they are stable in cases there are only very short training samples. On the other hand, since the discrimination between the authors is decided by a single metric (i.e., the similarity function) it is crucial to choose a stable and accurate function able to deal with multiple candidate authors, imbalanced and limited training sets.

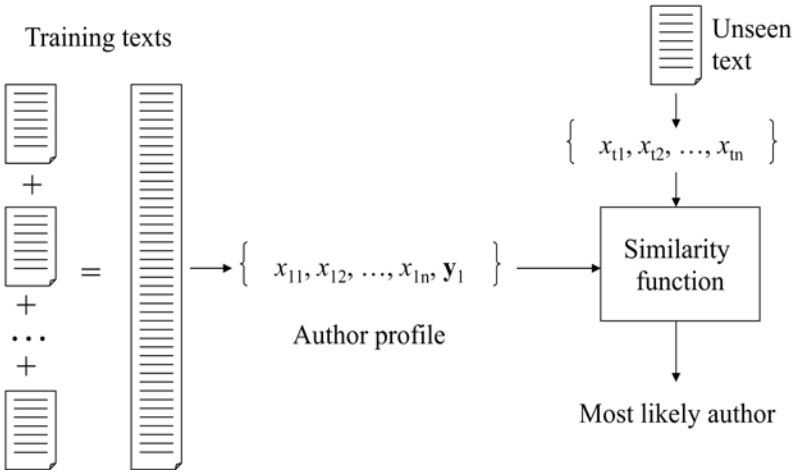

Figure 1. Architecture of the profile-based paradigm (for just one candidate author).

```
Input
C: set of candidate authors
D: training set
U: an unknown sample
n: n-gram order
L: profile size
d: similarity measure

CNG(C,D,U,n,L,d)
N_U=extract_ngrams(n,U);
N_U=sort_by_frequency(N_U);
P_U=get_profile(L,N_U);
foreach c ∈ C
      D_c=concatenate(c,D);
      N_c=extract_ngrams(n,D_c);
      N_c=sort_by_frequency(N_c);
      P_c=get_profile(L,N_c);
      S_c=similarity(d,P_U,P_c);
endfor
return argmax_c(S_c);
```

Figure 2. The CNG algorithm.

Common n-grams (CNG) is perhaps the most representative approach of the profile-based paradigm in author identification literature [12]. Moreover, this method and its variations have already produced very good results when applied to authorship analysis tasks in source code samples [2, 7] and natural language texts [13]. The main idea of this algorithm is described in the pseudocode of Figure 2. The profiles of the sample of unknown authorship and the concatenated samples of each author are extracted and then the similarity between these profiles estimates the most likely author. The profile of a sample is composed by the L most frequent n-grams found in this sample. This method has been proved to be very effective when it is combined with byte-level n-gram representations of source code [2, 7]. There are three significant parameters that should be tuned: n that is the order (length) of n-grams, L that is the size of the profile, and d that is the similarity function.

Previous studies on source code author identification have used the *Simplified Profile Intersection* (SPI) similarity measure which simply counts the number of common n-grams between two profiles disregarding any frequency information. This combination of CNG with the SPI measure is called Source Code Author Profile (SCAP) in some studies [2, 7]. Given two profiles X and Y, SPI is defined as follows:

$$SPI(X, Y) = |X \cap Y|$$

An alternative measure that has been proposed and tested in author identification and plagiarism detection of natural language texts [13, 14] is the *Asymmetric Normalized Relative Distance* (ANRD) defined below:

$$ANRD(X,Y) = \frac{\sum_{g \in X} \left(\frac{2(f_X(g) - f_Y(g))}{f_X(g) + f_Y(g)} \right)^2}{4|X|}$$

where $f_X(g)$ and $f_Y(g)$ are the normalized frequencies of the n-gram g in the profiles X and Y, respectively and $|X|$ is the size of profile X. The denominator guarantees that this function takes values in [0,1]. The lower it is the more similar are the two profiles. Thus, this dissimilarity function can be transformed to a similarity function by considering 1-ANRD(X,Y).

Note that ANRD(X,Y) is not symmetric since the sum of the nominator is calculated over the members of profile X that should always correspond to the sample of unknown authorship while profile Y should correspond to the concatenated training samples of a candidate author. In other words, it is assumed that X corresponds to a relatively short sample while Y corresponds to a relatively long sample. For this reason, this measure is very stable in case of skewed training sets where the size of the profiles of candidate authors may vary significantly. For instance, if we set n=4 and L=5,000 and for some authors there are just a couple of short samples in the training set, the size of the extracted profiles for these authors may be well below 5,000. It has already been used successfully to author identification and plagiarism detection tasks for natural language texts [13, 14].

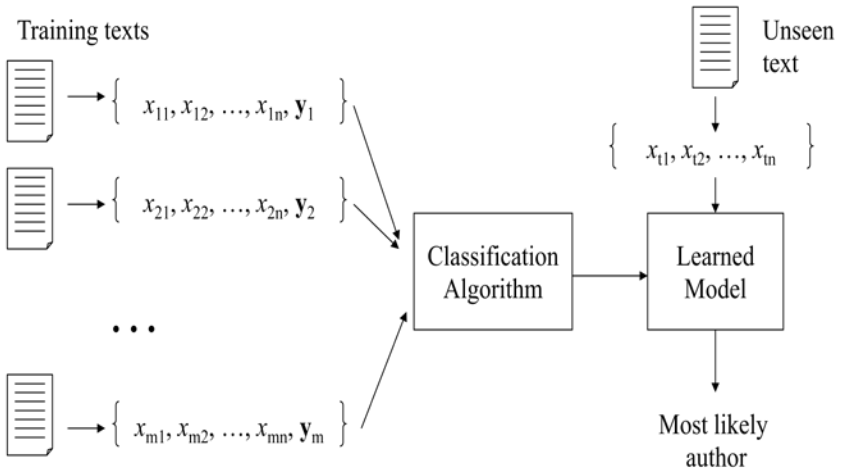

Figure 3. Architecture of the instance-based paradigm.

B. *Instance-based Paradigm*

The instance-based paradigm attempts to capture the style of individual samples [8]. It produces one separate representation per training sample and builds a classification model that can estimate the most likely author of a source code sample of unknown authorship. This procedure is illustrated in Figure 3. The machine learning algorithms usually exploited in the framework of this paradigm (e.g. SVM, neural networks, etc.) usually require multiple instances per class in order to be effective. Thus, in case there is only one long training sample for one author it should be appropriately segmented into shorter pieces. The instance-based methods can easily combine different types of features for representing source code. However, the representation of stylistic properties of very short samples may not be accurate. These methods are particularly affected by the class imbalance problem. Both the number of training samples and the size (in KBs) of the training samples per author affect the skeweness of the training set.

SVM is one of the most effective and well-known machine learning algorithms for text categorization tasks. In authorship attribution, it has been used in combination with character *n*-grams providing very good results [11]. Essentially, it is an instance-based method that is for each individual training text a representation is produced, usually based on the frequencies of the most frequent byte-level *n*-grams of the training corpus. The SVM algorithm can be used to learn the boundaries between classes (i.e., authors). The learned model is then applied to another sample of unknown authorship to guess the most likely author. In this paper, we used the LIBSVM implementation of this algorithm. Since the dimensionality is high (several thousands of features), the linear kernel has been used.

IV. EXPERIMENTS

A. *Data and Settings*

Since our aim is to study the class imbalance problem, we need an initially balanced collection, both in terms of number of samples and lines of code (or KBs) per author. By modifying the quantity of training set for specific authors of such a collection it is possible to produce multiple artificially imbalanced training sets. Unfortunately, the datasets used in previous studies do not fulfill these requirements since they are either too small or (more frequently) for some authors there is a very restricted number of samples available [2, 3, 4, 7]. Hence, we chose to build a new collection that is more appropriate for our experiments.

The source code samples used in this study come from the *JNode* project [1], a Java platform operating system. A few dozens of developers have contributed to this open-source project although most of the code was produced by a couple of programmers. The total size of this project is more than 200,000 lines of code. In order to group source code files according to the original authors we used the following information:

- The Apache Subversion (SVN) system provided the name of the committer for each file as well as the version of the file.
- The comments inside the file indicate the name of the developer.

In particular, we used source code files where the name of the committer and the developer was the same. Moreover, we only used the first version of each file to minimize the risk of reusing code from other developers. Since all source code files are parts of the same project this collection resembles the practical application of assigning parts of code to the contributors of a project.

To build a balanced dataset we first anonymized the samples by removing all comments that included author names and concatenated all samples per author into a single file. Then, we only considered authors who contributed at least 40KB of code in total. A set of 7 authors fulfilled this criterion while the minimum size of the concatenated files per author was 42KB. These files were down-sampled to 42KB and split into samples of 1KB each. Thus, the balanced collection we built consists of 7 candidate authors, and 42 source code samples of 1KB per author.

This collection of source code was split into training set (38 samples per author) and test set (4 samples per author). An

[1] http://www.jnode.org/

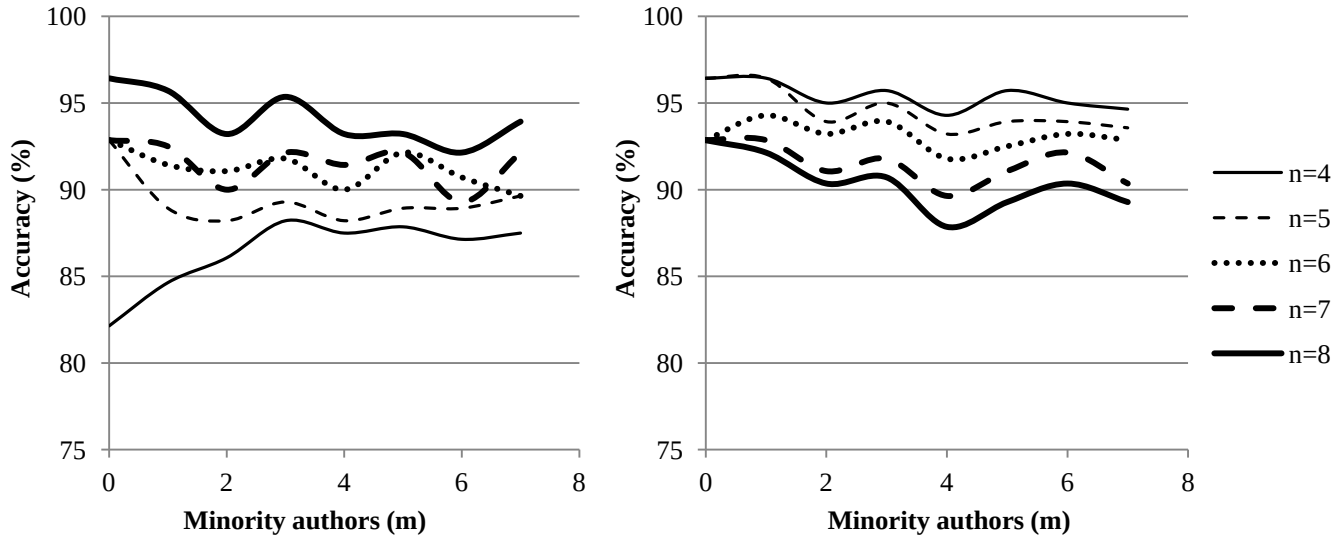

Figure 4. Comparison of the profile-based method (left, CNG with SPI) and the instance-based method (right, SVM) for low degree of class imbalance in the training set (r=30%) for different byte-level *n*-gram representations.

artificial case of class imbalance can then be produced by removing some training texts from some of the candidate authors. In other words, each skewed training set used in our experiments is a part of the initially balanced training set. There are two parameters that define the properties of each skewed training set:

- The number *m* of authors with reduced training samples. We called them *minority authors*. On the other hand, the authors with maximum number of training samples are called *majority authors*. We only consider cases where each candidate author may be either a minority or a majority author (i.e., there are no authors with less training samples than a majority author and more training samples than a minority author).
- The ratio *r* of training samples of a minority author that are removed from the training set with respect to the maximum number of training samples. This indicates the difference (in training samples) between the minority and the majority authors. We call this the *degree of class imbalance*.

For example, the case where *m*=2 and *r*=30% means that there are 2 minority authors and each one of them has 30% less training samples with respect to the full training set (i.e., 38 samples per author). The rest 5 authors have the maximum number of training samples (i.e., majority authors).

In our experiments, we vary *m* from 0 (when there is no minority author) to 7 (when there is no majority author) with a step of 1. Note that the boundary values *m*=0 and *m*=7 correspond to balanced training sets while the latter essentially represents the technique of down-sampling. Moreover, we examine three values of *r*: 30% (low degree of class imbalance), 60% (medium degree of class imbalance), and 90% (high degree of class imbalance). In each case, the test set remains the same and its distribution over the authors is balanced. Note that the test is always balanced and this is in accordance to the guideline introduced in [8], namely in author identification examined in the framework of forensic applications the training and test sets should not follow the same distribution. In other words, the availability of many training samples for one author should not increase the probability of being the author of a sample of unknown authorship.

In the presented experiments, the examination of each artificially imbalanced case (a combination of *m* and *r* values) was repeated 10 times, each time the specific minority authors and the removed training samples were selected randomly. In the experiments carried out in this study, the byte-level *n*-gram representation was used for quantifying the style of source code samples. For the CNG method the profile consists of the 5,000 most frequent *n*-grams (*L*=5,000). For the SVM model, any *n*-gram with frequency more than 3 (in the training samples) is included in the feature set. Preliminary experiments with different profile sizes and frequency thresholds did not affect the results drastically.

B. The Effect of Source Code Representation

First, we consider how the order of the byte-level *n*-gram representation affects the performance of the examined methods. We examined *n*-gram representations with *n*=4, 5, 6, 7, and 8 for both the CNG (with SPI similarity measure) and the SVM models in skewed training sets produced as explained in the previous section. Figure 4 shows the performance (i.e., classification accuracy averaged over 10 runs) of these models for *r*=30%. As can be seen, in general the performance of CNG improves when the order of the *n*-gram representation increases. This has also been noticed by [3]. Interestingly, the SVM model follows exactly the opposite pattern. Its performance drops when the order of the *n*-gram representation increases. The same patterns are obtained when examining other values of *r* (60% and 90%), not shown here for the sake of brevity.

This experiment indicates that CNG is better able to handle sparse features, namely long byte-level *n*-grams that may correspond to strings that include sequences of identifiers,

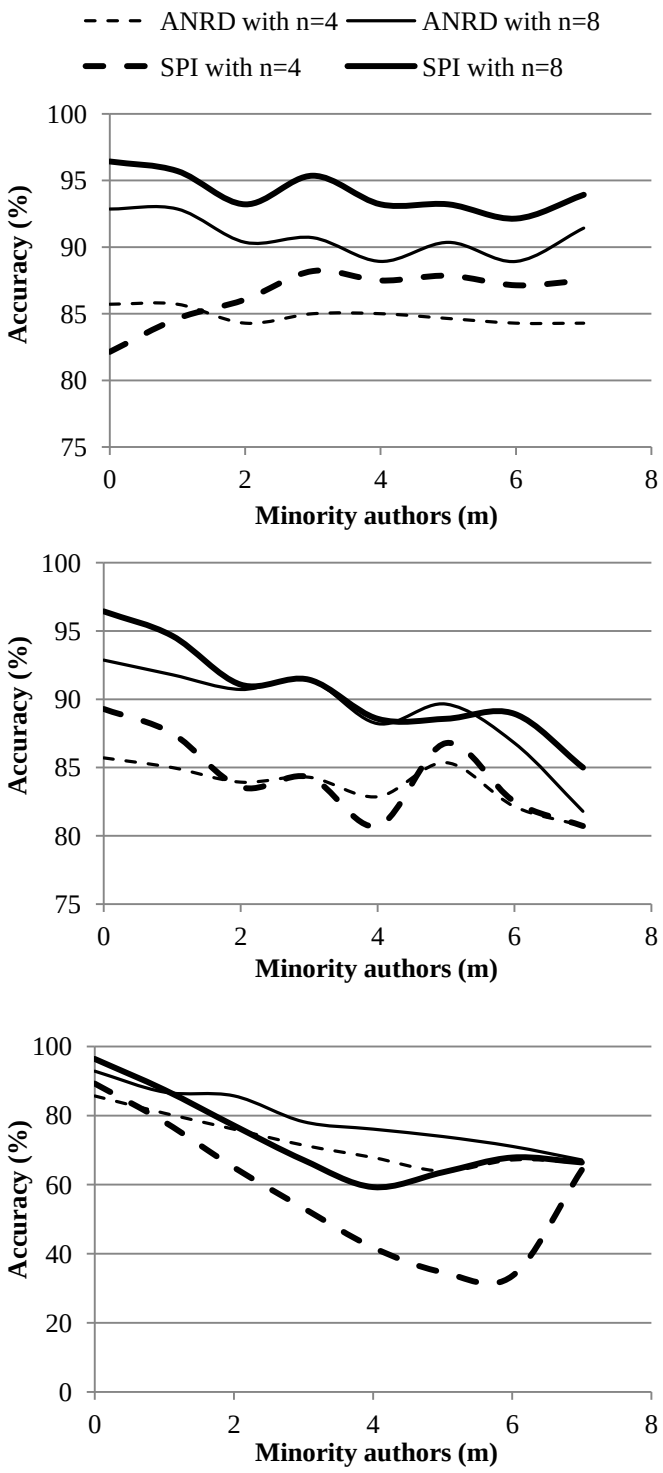

Figure 5. Comparison of the profile-based method (CNG) with the SPI and ANRD similarity measures for low degree (top), medium degree (middle), and high degree (bottom) of class imbalance in the training set.

operators, function names, etc. On the other hand, the SVM model can better handle short byte-level *n*-grams that may capture single identifier names, keywords, etc. or parts of that information. Note that short *n*-grams are more flexible and can capture the similarity of almost identical lines of code. For example, "int a[N];" and "int b[N];" have 2 common 4-grams while there is not any common 6-gram.

C. The Effect of Similarity Measure

In the next experiment, we examined the CNG model in combination with two similarity measures: SPI and ANRD. Figure 5 depicts the performance of these models in skewed datasets for byte-level *n*-gram representations with n=4 and n=8 while the degree of class imbalance is 30%, 60%, or 90%. As shown in the previous experiment, the performance of the models based on 4-gram and 8-gram representations can be considered as indicative lower and upper boundaries, respectively, of the performance of the CNG models. For low degree of class imbalance (r=30%), the SPI measure clearly outperforms ANRD. For medium degree of class imbalance (r=60%) the performance of SPI and ANRD models is comparable although SPI seems to be significantly better in balanced (m=0 and m=7) and near-balanced datasets (when there is just one minority author or just one majority author). For high degree of class imbalance (r=90%) the ANRD models are more stable and outperform SPI models when there are at least 2 minority authors.

SPI seems particularly vulnerable when there are just a few majority authors. In those cases the majority of the unseen samples tend to be classified under the majority authors because they have considerably larger profiles than the rest of the authors. Hence it is more probable to find common *n*-grams in the profiles of a majority author and an unknown sample. The use of ANRD avoids this problem.

D. Profile-based vs. Instance-based Author Identification

Finally, we compared the performance of CNG (with ANRD similarity measure) and SVM models for varying degrees of class imbalance and different byte-level *n*-gram representations. Again, the models with n=4 and n=8 can be considered as indicative lower and upper boundaries, respectively, for the performance of the CNG models and the opposite, that is lower and upper boundaries for the performance of the SVM models. The results shown in Figure 6 demonstrate that for low degree of class imbalance SVM is clearly the best performing method. Both CNG and SVM remain practically stable despite the reduction of the training set for some candidate authors. The under-sampling technique (i.e., it corresponds to the case where m=7) seems to be a good solution when half or more of the candidate authors are under-represented (minority authors).

In the case of medium degree of class imbalance, CNG is more stable but SVM still provides the best performing model. When there are enough minority authors (m>2) the under-sampling technique seems to be a good choice for SVM but it is not effective for CNG.

For high degree of class imbalance CNG clearly outperforms SVM. In that case, the under-sampling technique provides very poor results for the SVM models. It is not a good option for CNG either since the performance of CNG drops quasi-linearly by increasing the number of minority authors. The classification accuracy of CNG drops from 92% (m=0) to 67% (m=7) despite the fact that the training set has been drastically reduced from 38 samples per author to just 4 samples per author. On the other hand, the performance of SVM drastically drops below 40%.

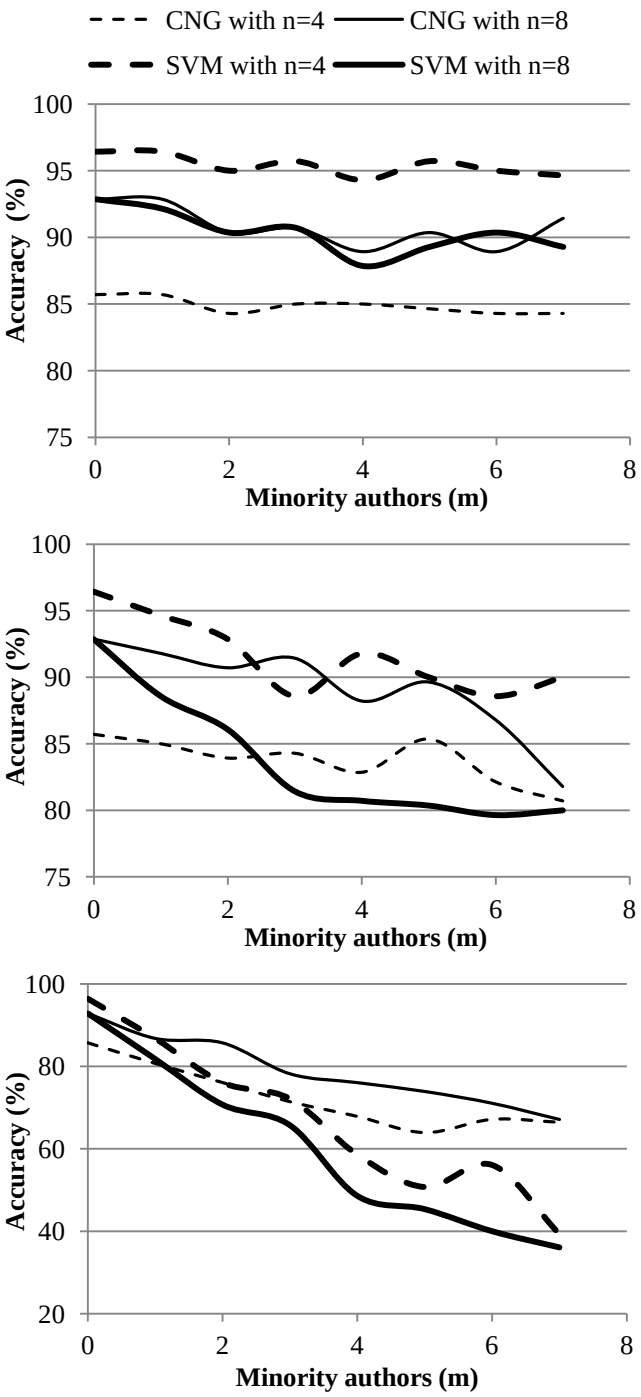

Figure 6. Comparison of the profile-based method (CNG with ANRD) and the instance-based method (SVM) for low degree (top), medium degree (middle), and high degree (bottom) of class imbalance in the training set.

This experiment indicates that the profile-based methods are better able to handle both balanced datasets of limited size (i.e., with just a few training samples per author) as well as highly skewed datasets in comparison to instance-based methods. On the other hand, instance-based methods are more appropriate when the training set is slightly-skewed.

V. CONCLUSIONS

In this paper, we presented a detailed experimental study of the class imbalance problem in the source code author identification task. We examined two different methods of author identification representing the two basic paradigms, that is the profile-based paradigm and the instance-based paradigm. Based on a collection of source code samples and different degrees of class imbalance as well as different number of minority authors we demonstrated the performance of these author identification methods under a variety of cases of skewed training sets.

Byte-level *n*-grams offer a computationally simple and programming language-independent approach to represent the stylistic properties of code and have already been used by previous studies with very promising results. We examined the author identification methods with different orders of byte-level *n*-grams and an interesting conclusion is that although the profile-based method performs better with long *n*-grams (something already indicated by previous studies), the instance-based method is more effective with short *n*-grams. This pattern is not affected by the degree of class imbalance in the training set.

One main conclusion of this study is that the instance-based method (SVM) is more accurate in balanced or nearly-balanced cases (when the degree of class imbalance is low). In the latter case, both methods are quite stable since their performance is not heavily affected by increasing the number of minority authors. For medium degree of class imbalance, the instance-based method continues to be superior but its stability is now considerably decreased. Hence, when the training samples of the most under-represented authors are at least a half of the most over-represented authors, the SVM model seems to be the better option. On the other hand, in cases with high degree of class imbalance, the profile-based method (CNG) is more accurate and more stable in comparison to SVM. Thus, when there are some very under-represented authors in the training set, the profile-based method seems to be the best solution. In addition, the under-sampling technique seems to be a good option for both SVM and CNG when at least half of the candidate authors are under-represented in the training set and the degree of class imbalance is low or medium. However, for highly-skewed datasets, it is not appropriate for both methods.

The comparison of two similarity measures (SPI and ANRD), proposed in previous author identification studies to be used with the CNG method, demonstrated that SPI is better in cases of low degree of class imbalance while ANRD is better in cases of high degree of class imbalance. For medium degree of class imbalance the two approaches are practically equally effective. It should be noted that SPI is particularly vulnerable when there are just a few majority authors and the degree ofclass imbalance is high. In those cases the unknown samples tend to be classified under the majority authors simply because they have larger profiles than the rest of the authors. However, CNG with SPI is still more accurate than SVM in those cases.

An interesting future work dimension is to explore the usefulness of methods that produce synthetic data to re-balance the training set [10, 11]. Moreover, alternative code representation methods [3] could be tested.

REFERENCES

[1] I. Krsul and E.H. Spafford, "Authorship analysis: Identifying the author of a program", Computers and Security, 16(3), pp. 233-257, 1997.

[2] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas, "Effective identification of source code authors using byte-level information", in *Proc. of the 28th International Conference on Software Engineering*, 2006, pp. 893-896.

[3] S.D. Burrows, *Source Code Authorship Attribution*, Ph.D. Thesis, RMIT University, 2011.

[4] S.G. MacDonell and A.R. Gray, "Software forensics applied to the task of discriminating between program authors", *Journal of Systems Research and Information Systems*, 10, pp. 113-127, 2001.

[5] H. Ding and M.H. Samadzadeh, "Extraction of Java program fingerprints for software authorship identification", *The Journal of Systems and Software*, 72(1), pp. 49-57, 2004.

[6] R.C. Lange and S. Mancoridis, "Using code metric histograms and genetic algorithms to perform author identification for software forensics", in *Proceedings of the Ninth Annual Conference on Genetic and Evolutionary Computation*, 2007, pp. 2082-2089.

[7] G. Frantzeskou, S.G. MacDonell, and E. Stamatatos, "Source code authorship analysis for supporting the cybercrime investigation process", In Chang-Tsun Li (ed.) *Hndbook of Research on Computational Forensics, Digital Crime, and Investigation: Methods and Solutions*, pp. 470-495, 2010.

[8] E. Stamatatos, "A survey of modern authorship attribution methods", *Journal of the American Society for information Science and Technology*, 60(3), pp. 538-556, 2009.

[9] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study", *Intelligent Data Analysis*, 6(5), pp. 429-450, 2002.

[10] N. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique", *Journal of Artificial Intelligence Research*, 16, pp. 321-357, 2002.

[11] E. Stamatatos, "Author identification: using text sampling to handle the class imbalance problem", *Information Processing & Management*, 44(2), pp. 790--799, 2008.

[12] V. Keselj, F. Peng, N. Cercone, and C. Thomas, "N-gram-based author profiles for authorship attribution", in *Proc. of the Pacific Association for Computational Linguistics*, 2003, pp. 255-264.

[13] E. Stamatatos, "Author identification using imbalanced and limited training texts", in *Proc. of the 4th International Workshop on Text-based Information Retrieval*, 2007, pp. 237-241.

[14] E. Stamatatos, "Intrinsic plagiarism detection using character n-gram profiles", in *Proc. of the 3rd Int. Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse*, 2009.