Michal Tereszkowski-Kaminski*, Sergio Pastrana, Jorge Blasco, and Guillermo Suarez-Tangil

Towards Improving Code Stylometry Analysis in Underground Forums

Abstract: Code Stylometry has emerged as a powerful mechanism to identify programmers. While there have been significant advances in the field, existing mechanisms underperform in challenging domains. One such domain is studying the provenance of code shared in underground forums, where code posts tend to have small or incomplete source code fragments. This paper proposes a method designed to deal with the idiosyncrasies of code snippets shared in these forums. Our system fuses a forum-specific learning pipeline with Conformal Prediction to generate predictions with precise confidence levels as a novelty. We see that identifying unreliable code snippets is paramount to generate high-accuracy predictions, and this is a task where traditional learning settings fail. Overall, our method performs as twice as well as the state-of-the-art in a constrained setting with a large number of authors (i.e., 100). When dealing with a smaller number of authors (i.e., 20), it performs at high accuracy (89%). We also evaluate our work on an open-world assumption and see that our method is more effective at retaining samples.

Keywords: Authorship Attribution, Underground Forums, Language Selection, Code Clone Detection

DOI 10.2478/popets-2022-0007
Received 2021-05-31; revised 2021-09-15; accepted 2021-09-16.

1 Introduction

Underground forums allow users interested in illicit activities to interact with and learn from their peers [32, 35, 40]. In recent years these forums have also become improvised marketplaces, where users trade and share illicit products and services [11]. These include both physical (e.g., drugs or weapons) and virtual (e.g., malware or exploits) items. As a result, these forums lower the entry barrier into cybercriminal activities and have become a key area of investigation for industry, academia, and law enforcement [33]. A popular type of product found in these forums is software, e.g., tools to run cyber attacks, malware, or video-game hacks and cheats [46]. Users of underground forums often provide the source code of such tools [40]. In some cases, this is a result of them willing to share their knowledge to increase their reputation [36]. In others, this is due to users asking for advice, similar to other question-and-answer sites.

The illegal and illicit nature of the activities carried on in these underground forums motivates users to use pseudonyms. This allows them to preserve their privacy and also to hide their true identity, e.g., to hinder the investigation by law enforcement and security practitioners [31]. Accordingly, stylometry analysis has emerged as an increasing area of research aimed at the actual identification of these users, both from their language [4] and from the type of content posted [52].

Code stylometry is the practice of extracting stylistic meta-information from source code. It might allow for example to find whether different accounts belong to the same person or to identify different users from the same group or gang. So far, existing solutions [16, 21, 45, 51] are typically tested on curated datasets such as the Google Code Jam (GCJ) dataset [23], or code from open-source repositories such as GitHub. While these datasets contain enough representative data (authors and samples) to perform classification experiments, they do not represent the kind of data that can be obtained in the wild from underground forums or other online communities, as this data is usually unbalanced, unstructured and noisy. Accordingly, a potential, yet largely-unexplored area is the analysis of the stylometry of the source code shared in these dynamic and unpredictable real-world settings. Caliskan et al. [21] made inroads into this question by studying publicly available code repositories on GitHub. They achieved 50%-60% accuracy, showing that the state-of-the-art methods are unable to achieve 95+% accuracy in the real-world setting of collaborative GitHub repositories. GitHub is a popular platform, and it normally

***Corresponding Author: Michal Tereszkowski-Kaminski:** King's College London, E-mail: michael.tereszkowski-kaminski@kcl.ac.uk
Sergio Pastrana: Universidad Carlos III de Madrid
Jorge Blasco: Royal Holloway, University of London
Guillermo Suarez-Tangil: IMDEA Networks Institute and King's College London

includes well-formatted. However, it is unclear to what extent these observations apply to underground forums.

In this work, we study the applicability of state-of-the-art authorship attribution techniques in data gathered from various underground forums and show that they suffer several limitations which renders them useless in this environment. Accordingly, we propose a methodology specifically designed to deal with the idiosyncrasies of code shared in these settings. Specifically, in this work we present the following contributions:

1. We identify particular challenges of code stylometry applied on snippets shared in underground forums (§2), and show that state-of-the-art code stylometry techniques, while working well for curated datasets, are not appropriate in these settings (§3). We identify possible reasons why accuracy drops.
2. We design and implement a methodology to perform attribution by means of code stylometry in underground forums. As a key novelty, it incorporates a statistical mechanism to evaluate the classification quality and discard unreliable samples, increasing the accuracy and confidence of the attribution (§4).
3. We provide empirical results under closed-world and open-world assumptions. We show the benefits (i.e., doubling accuracy) of applying our methodology to incomplete and unstructured data, while still obtaining high effectiveness on curated datasets (§5). We also present two case studies that show how our methodology can assist in the identification of users posting across different forums (§6).

We discuss our findings and the limitations of our methodology in §7. We present a detailed breakdown of related work in §8. We release our tool and provide further instructions on how to reproduce our experiments: https://github.com/MichaelTK/code-stylometry-UF. We address the ethical issues that arise as a result of our work in Appendix D.

2 Problem Statement

In recent years, online underground forums have turned into improvised online markets for various illicit products and services [5, 40, 48]. They are also used by newcomers to learn and initiate in deviant topics [24, 36]. Consequently, understanding and measuring their underlying economy has attracted considerable attention in academia [8, 43, 46]. The cybercriminal underground markets have also become an interesting source of information for law enforcement [33] and security researchers [31]. One domain where code stylometry can contribute is in the attribution of software produced for malicious purposes, like malware [40] or exploits [5]. Malware development has evolved over time, and current deployments are similar to complex projects [17]. Thus, writing malware requires effort and expertise, which can be found in underground communities [35].

Code stylometry consists of extracting stylistic information from program source code and using it to identify the programmer [15, 22]. It has potential privacy implications since it might be used to de-anonymize programmers by analyzing their source code [16]. It can be also used by law enforcement and security investigators in the attribution of cyberattacks.

Analyzing code gathered from underground forums brings particular challenges. First, a given code sample might be written by multiple authors, all contributing their unique style to it. This can be addressed by focusing on function units [21], assuming that each function will have only been written by one author. Second, the ground truth is unreliable: the same developer might operate different accounts, and the same account might be operated by different users. Third, code is often incomplete and might contain syntax or grammar errors. This means that analysts will sometimes deal with code that does not compile, thus affecting the parsing and extraction of features often used for attribution [16, 21, 49]. Finally, the nature of the code is unknown: whereas code collected from programming contests such as GCJ [23] aims at solving a well-defined problem, code samples gathered from underground forums encapsulate a wide range of purposes, including licit (e.g., defensive tools or programming contests) or illicit (e.g., malware or game cheats) activities. Changes in the nature of the code can affect the accuracy of the attribution [13].

To better understand these challenges, the next section presents an empirical study of the effectiveness of code stylometry on different settings.

3 Challenges in Code Stylometry

In this section, we offer a baseline experimentation using state-of-the-art code stylometry techniques over three different datasets. Our experimentation shows how these techniques are under-equipped to deal with code snippets posted in underground forums.

3.1 Datasets

Many of the previous works performing program authorship attribution with source code [6, 16, 45, 51] leverage curated datasets such as the programming contest Google Code Jam (GCJ) [23] or GitHub (GH) repositories. We use these two datasets as a baseline to test the effectiveness of existing tools. The GCJ dataset contains code from an international competition of algorithmic problems. We also scrape 100 malware-oriented repositories from GitHub following the methodology from [6]. We take into account only repositories with a single contributing user account. However, this does not guarantee that this contributor is a single author, since the repository could include external libraries or multiple authors commit code using the same account.

In our experiments, we also use data from the CrimeBB dataset, which contains data scraped from various underground forums [36]. The data is provided by the Cambridge Cybercrime Centre[1] and is available to academic researchers under a legal agreement designed to ensure ethical use. We dub the dataset of code snippets extracted from these forums as UF (Underground Forums).

Overall, the UF dataset is composed of 90,085 samples of over 19,845 authors from 7 underground forums, with 99.5% of the code snippets belonging to 2 of these forums. As concerns the GCJ dataset, we focus our study on code related to the submissions made in one year, totaling 31,008 code samples over 5,830 authors. The GH dataset consists of 5,639 code samples over 100 repositories (i.e., 100 authors).

3.2 Motivation Example

We start by reproducing the results of previous work [16] and establish a baseline for the effectiveness of the state of the art on curated data. We select this as our starting point because its dataset has been widely used by the community as a benchmark [6, 16, 45, 51] and because they were the first ones to introduce the Code Stylometry Feature Set (CSFS) [16, 21]. We first apply the CSFS to the GCJ dataset. In particular, we sort the authors based on the number of samples provided and get the top 100. From these, the minimum number of samples is 8 and the maximum is 18 (with a standard deviation of 1.7). We produce feature vectors for each sample

1 www.cambridgecybercrime.uk

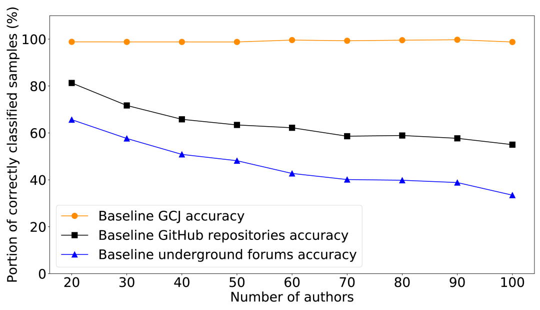

Fig. 1. Attribution accuracies in pre-methodology GCJ, underground forums and GitHub code.

using CSFS, and use a Random Forest classifier and 5-fold cross-validation. The use of all features produces upward of 95% accuracy in a closed-world attribution task of 100 authors. In this setting, samples are informative enough and the classifier is able to learn stylistic information, independently of the number of authors (see GCJ baseline results in Figure 1).

We proceed in a similar fashion and apply the CSFS to the GH dataset, by sorting based on the number of samples provided. Figure 1 shows that the accuracy of the classifier decreases as more authors are considered, but remains above 60% in all settings. Finally, we apply the same learning setting to the UF dataset.[2] In this case, the per-sample out-of-the-box classification accuracy is considerably lower than both GCJ and GH (see UF baseline results in Figure 1). For this attribution task, we select 100 of the user accounts with the largest number of C/C++ code snippets posted. We randomly select 15 samples per author and perform a 5-fold cross-validation authorship attribution task. The accuracy of the attribution in UF ranges from 65.6% (using 20 authors) to 33.4% (using 100 authors), which is significantly worse than our baselines (ranging from 30% to 60% for GCJ, and close to 20% for GH in most settings).

3.3 Factors Affecting Attribution

Our initial results show that, while current stylometry techniques work well with curated datasets, there is a degradation when dealing with less curated datasets reaching a point of very poor performance when dealing with unstructured data gathered from underground forums. We perform a comprehensive analysis of the properties of the GCJ dataset and contextualize it with the

2 We provide details on how data is preprocessed in §4.2.

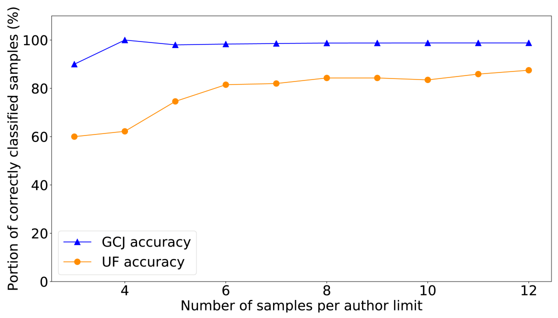

Fig. 2. Attribution accuracy vs samples per author in learning.

characteristics of the data we find in other settings like GitHub and underground forums.

Due to space constraints, we report the characterization and comparison of the three datasets in Appendix A. Our analysis reveals the following issues that might affect the performance of a system trained to perform authorship attribution (for a full evaluation of the effects of these factors consult Appendix B.):

1. *Number of samples per author.* The median of samples per author in GCJ is 6, and the standard deviation is 1.4. In contrast, the median of samples per author in UF is 2 but the standard deviation is 16.5. For GH, the median number of samples is 16 with a much higher standard deviation (119.1). In the GCJ dataset, most authors submit the same number of solutions (samples). For GH, a single malware sample will be composed of several source code files. Using the GCJ dataset, where other conditions are kept optimal, the number of samples per author does not have a significant effect on attribution accuracy past 5 samples per author. This is likely due to the wealth of information gathered from each sample, their median length being 68 LoC, which is relatively long compared to UF. In UF, authors possess greatly varied amounts of code, and thus the dataset is highly imbalanced. The effects of these discrepancies on attribution is shown in Figure 2.
2. *Type of programs.* In GCJ, most authors solve the same problems,[3] and thus the code is developed for the same type of programs. In GH, all programs are malware, though the functionality is not restricted to a specific type. By contrast, code from underground forums may be related to a completely unrelated problem or task.
3. *Sample size.* In GCJ, the solutions proposed have a similar complexity (in terms of LoC). In UF and GH, there is a much greater variation in snippet length (standard deviation of 105.1 and 4,047 respectively). In addition to this, code samples for GCJ and GH are generally longer with medians of 68 LoC for GCJ, 158 LoC for GH and 3 for UF.
4. *Quality of the ground truth.* Different from the GCJ dataset, labels in the UF dataset are unreliable. In the case of GH, we know the repositories have been contributed by a single user, but the ground truth is dubious since these repositories might include code from third-parties. While precautions can be taken by removing code clones, it is uncertain how much noise remains present in the ground truth of the samples.
5. *Completeness* of the code. Code from GCJ must compile and execute (a condition of the contest). Also, for GH we assume that code will at least compile. Code shared in online communities does not necessarily need to compile and may contain syntax errors as the main purpose for sharing the code might not be its direct execution. Since code stylometry relies on both syntactic and semantic features, this clearly affects the attribution accuracy.

3.4 Deconstructing the GCJ Dataset

We next explore how the different factors in §3.3 affect the performance of existing stylometry techniques by "poisoning" the GCJ dataset. We limit this analysis to GCJ as it is the one with most distinctive characteristics when comparing it with UF. Also, as mentioned before, the GCJ dataset offers the best scenario for code stylometry analysis, and data from underground forums differ from such ideal scenarios. In particular, we run the learning setting used in our motivation example (§3.2) while we deconstruct the GCJ dataset to an extent similar to the one we observe in UF.

We refer the reader to Appendix B for more details on how this is done. We next summarize our findings, which are then used to inform our thresholds when dealing with UF in §5.2:

Number of samples per author. To analyze the effects of dataset imbalance, we modify the GCJ dataset by selecting the top 20 authors (with most samples) and limiting the number of samples per author used for training. Our results show a stabilization in performance when at least 5 samples are available for training and testing, using 5-fold cross-validation. It is worth men-

3 The number of problems is limited from the beginning to 9.

tioning however that performance improves past this number of samples when using a degraded dataset (samples gathered from underground forums).

Sample size. We modify the GCJ dataset by reducing the number of LoC per sample. We observe that attribution accuracy follows a linear positive relationship with sample length until around 90 LoC, where accuracy levels off at its peak of 98%.

Quality of the ground truth. Finally, we modify the dataset by randomly mislabeling samples. Under these conditions, changes in the ground truth noise have an expected negative linear relationship (Figure 3) with the accuracy of authorship attribution of samples from 98% to 2% for 80% of the ground truth poisoned.

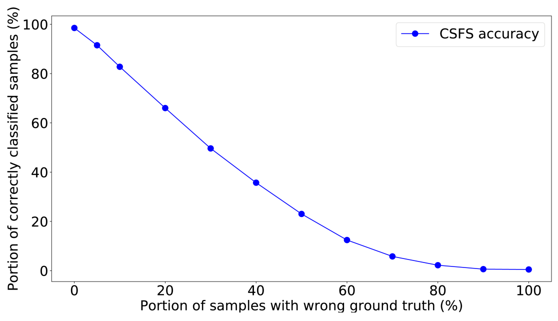

Fig. 3. Attribution accuracy variable with the percentage of mislabeled samples in GCJ.

3.5 Lessons Learned

The analyzed differences might have an important effect on the performance of a system trained to perform authorship attribution. For instance, when looking at the type of programs we argue that the low variance in the GCJ dataset (it is limited to 9 different problems) removes variance in context from the samples, helping the classification system to focus on the stylistic differences between authors. We are unable to engineer the GCJ dataset to vary the context of the samples within. The GCJ contest offers a very limited number of problems for the code to solve, and in such circumstances, we argue that the classifier will easily capture artifacts dependent on the programmer's style. It could be argued that this will negatively impact attribution in domains with less constrained programming contexts, however, this is not clear to be the case, as varied context can also mean less subtle code-wise variation between the samples of different programmers. The CSFS was designed with programming style in mind so it may not be severely affected as several authors perform the same task. However, it may not be able to yield accurate attribution when dealing with authors that perform a wide range of different tasks. Our paper seeks to understand how all these differences produce a number of nuanced trade-offs when performing authorship classification.

Our experimentation with poisoned versions of the GCJ dataset indicates that current approaches have limitations with datasets where the ground truth is unreliable.For instance, out-of-the-box experiments with GH, while producing better results than out-of-the-box UF experiments (around 20% better accuracy), do not approach the accuracy of experiments on GCJ (around 20% to 50% worse). GH has a much greater wealth of data for each author label, but the ground truth can not be guaranteed to the same level as in GCJ. As in UF, users in GH can use the platform to share code that belongs to someone else.

This is a recurring issue for any supervised classification task, and learning in the presence of a "polluted ground truth" [3] is an active area of research [3, 18, 19, 26, 30]. In the UF domain, the code snippets posted by a given author could have been (partially) implemented by another author. While several efforts have been made to detect multiple accounts in social media data, such as posts from underground forums [4, 34, 38, 52], having reliable ground truth (i.e., accurate labels for all posts that belong to an author) remains an open challenge.

In order to deal with the factors we identify that affect the attribution, we introduce a novel classification framework that fuses our learning pipeline with *Conformal Prediction* [47]. This generates predictions with precise levels of confidence. To the best of our knowledge, we are the first to show that Conformal Prediction theory can be used to improve code stylometry in settings where obtaining a clean and curated dataset is challenging.

4 Methodology

This section presents the methodology we use to perform authorship attribution in underground forums.

The purpose of our methodology is twofold. First, we devise a method to perform source code authorship attribution that can also apply in constrained environments. As a byproduct, we develop a pipeline to assess how state-of-the-art tools perform in data gathered from real-world forums which serve as places to exchange code. We design our methodology as a general pipeline that is evaluated (§5) in two operational settings: one where we assume a closed-world problem and another where we test an open-world problem.

4.1 Overview

Our methodology is illustrated in Figure 4. We adapt the standard phases of a machine learning pipeline to overcome the issues of analyzing underground forums. The phases, along with the main changes we implement are as follows:

1. **Pre-processing**. First, we extract all code snippets and identify their programming language. We select those samples written in the languages we are concerned with and discard the others. At the core of our method, we perform code clone detection to identify sources of plagiarism and use it to curate the ground truth.
2. **Feature representation**. The next set of steps relate to the extraction of features and the identification of style. To increase the granularity of the samples and ease the removal of meaningless code, we split the snippets into their constituent functions. We then compute a feature vector for every data sample (function) extracted from the code. This is based on the Abstract Syntax Tree (AST) of the function.
3. **Learning**. Finally, we classify each individual sample to perform authorship attribution, perform thresholding of samples based on Conformal Predictor confidence, and propose an ensemble method to provide a unified label for every code snippet (set of functions).

We next explain each of these phases and provide details of their implementation.

4.2 Pre-Processing

The goal of the pre-processing phase is to extract code snippets from the posted content in the forums, select those using C/C++ language, to extract the functions from these snippets, and filter out code that decreases the quality of the resulting dataset (e.g. code clones or short samples). The goal is to produce a dataset in line with the attributes of a curated dataset such as GCJ.

Post extraction. We use the CrimeBB database (described in §3.1) to perform the initial data collection. In particular, we scrape all posts in 7 forums and extract snippets that are enclosed in-between specific <*CODE*> tags. However, we observe that these tags are often used for purposes other than posting code, such as enumerating steps to perform when creating a hack or sharing lists of emails. To confirm this, we randomly sample 100 posts to study the type of content posted within code tags. In general, we see that authors abuse code tags to format other types of content in their posts, which we dub *un-parseable code*. However, we see that overall about half of the snippets posted are actual source code. Thus, the next step is the identification of actual code, by looking for the specific programming language.

Language selection. We combine a data-driven approach to guess the programming language, together with a set of heuristics tailored to distinguish natural language from code. We first leverage Guesslang [50], a machine learning model trained with 1M unique source files in 30 different languages. We run a prediction task per snippet to identify those samples that contain C/C++ code. We then run a second step that filters out posts that resemble a program in C/C++ but does not actually contain *parseable code*, like for example raw chat logs, item lists (e.g. URLs, MD5 hashes, or emails), configuration files or another sort of metadata.

To filter out samples with *un-parseable code*, we use three heuristics looking at *natural language*, *URLs* and *size of code* as we explain next. These heuristics use thresholds, which are set by conducting manual experimentation over a random subset. While such heuristics are tailored to the UF dataset, they can be adapted to the particularities of other datasets.

Natural language. All the forums studied are in English, so natural language words should contain characters from the Latin alphabet. However, we observe that this is not necessarily the case, e.g. due to the use of emojis or specific jargon. Thus, we account for the number of words from each sample that contain non-Latin characters and filter the sample out if such a number is below 50% of the total words in a sample.

Lists of URLs. This heuristic filters content where the <*CODE*> tag is used to post lists of URLs. First, we use a regular expression to identify URLs, identifying "http://", "https://", "www" prefixes. Second, since we observe cases where URLs are embedded in *parseable code*, we set a conservative threshold: if more than 70% of the lines in a sample contain matching strings, we

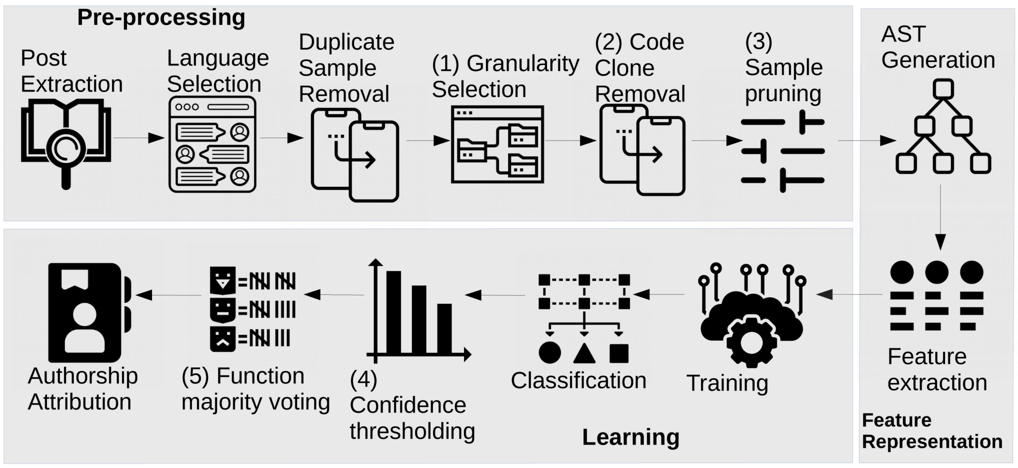

Fig. 4. Methodology outline.

remove the sample, assuming it is a list of URL rather than a code snippet. Experimentally this proved to produce the most acceptable compromise between allowing noisy URL lists and removing legitimate C/C++ code which contains multiple URLs.

Code size. The last heuristic removes short samples that are not informative. Concretely, we remove samples that are shorter than 20 characters. Samples of this size are either *un-parseable code*, or if *parseable*, these are too short to extract meaningful features.

Duplicate sample removal. We detect and remove duplicate samples, leveraging the Hamming distance. Concretely, if less than 1% of the characters in two samples are different, including white-spaces, tabs, and other such structural indicators, both samples are removed from the corpus since they have a high proportion (i.e. 99% or more) of common code. As such we remove samples for which the UF ground truth is contradictory.

Granularity selection. One of the contributions of our methodology is to parse code and split it into functions to conduct the stylometry analysis. Using functions rather than code snippets improves the attribution accuracy as we see later in §5.2).

In order to split into functions, we parse the samples with Joern [27], a C/C++ parser that allows identifying function definitions in a source code file. We then extract them into their own source code files. Joern constructs code property graphs from these files, from which we get the indexes of function identifiers. We then use these line indexes to parse from them to the end of the function definition, using a stack of curly braces to identify the start and end of the function definition construct.

Function clone removal. Similar to code clones, we remove function clones using the Hamming distance. If less than 1% of the characters in two functions are different, these are flagged as clones and removed from the corpus. In this way, we remove functions that might have been copied entirely, and for which ground truth is dubious. However, such a low threshold keeps functions which, even if originally copied from another source, have been somehow modified, introducing the new author's stylistic information.

Sample pruning. This stage aims at filtering out samples that are not informative enough. In particular, we set a minimum number of LoC that every sample must contain. Although the higher the LoC the better, the nature of underground forums, offering relatively few samples per author for all but the few most active ones, means that there might not be enough source code samples to conduct the analyses. Additionally, the fact that each author's submitted code snippets are further split into functions, means that there are very few code samples which are long enough. On the one side, focusing on authors who share larger samples, and more often, helps to analyze well-established members of the forums, and also allows to increase the accuracy and confidence of the attribution task. On the other side, low-interaction users are harder to de-anonymize and are generally users with lower impact on the community [35]. Thus, we discard those authors for which do not have a sufficient number of samples and maximize the LoC per sample. This is a common practice done by other related works in authorship attribution [4, 15, 44]. We analyze the operational settings that can be applied in the UF dataset in §5.1.

4.3 Feature Representation

Abstract Syntax Tree generation. We obtain a representation of each sample based on syntactic structures of the code. For this, we generate the Abstract Syntax Trees (AST) of every sample. Our implementation leverages Joern, which supports fuzzy parsing. As such, AST can usually be generated from code samples which would not normally compile during a standard build, a necessary feature when dealing with code from unpredictable sources like posts from underground forums.

Feature extraction. We evaluate a wide range of features from the CSFS [16]. Concretely, we obtain Lexical and Layout features from raw code fragments and Syntactical features from the AST. *Lexical* features include the number of parameters passed to functions or the unique word unigram term frequency (i.e., terms such as '`if`', '{' or '='). *Syntactical* features stem from the interpretation of the code and they capture properties of the program. These language-dependent features are able to represent the code at a level of syntactical abstraction. *Layout* features deal with the structure of the code such as the author's tendency to use tabs instead of newline characters etc. All features are then transformed into feature vector representations of each sample.

4.4 Stylometry-Based Learning

Training & classification. We perform 5-fold cross-validation using a random forest classifier. We use 5-fold cross-validation due to the scarcity of samples we have in our possession, allowing training and testing on the same samples. The minimum number of samples per author label we use is 5, as this was experimentally found to be the minimum which still produces optimal accuracy (Figure 2) in a curated dataset like GCJ, and results in a sufficient number of authors for a 100-author attribution task. We chose a random forest algorithm since it has been successfully used in prior works using the same feature set for the purposes of authorship attribution [16, 21].

Confidence thresholding. In this step, prediction confidence is used to reject samples which the classification pipeline is unable to classify with a satisfactory degree of certainty. This step prevents the pipeline output presented to the analyst from including predictions which are the classifier's "guess" rather than informed prediction. As discussed in §2, this is a necessary step when dealing with data from underground forums, where the ground truth is unreliable. Thus, only samples for which the classifier is more confident pass to the following step of the pipeline.

The methodology proposed admits any kind of statistical method for calculating prediction confidence, such as the calibration curves based on probabilities used in previous work [21]. As a novelty, we use a Conformal Predictor (CP) [47] for this purpose. This technique uses a predictive distribution function to assess how well a sample fits into a class, and it is capable of producing exact values of confidence in new predictions. Based on the confidence values produced, we reject samples which fall below a confidence threshold θ, as these samples have a high probability of belonging to another author. This usually happens when the suspect set does not have enough differentiating features to separate them apart. Experimentally, past this threshold we obtain severely diminishing returns on accuracy improvement while rejecting most of the samples in our dataset.

This method is more robust than those used in related work [21] because it converts probabilities based on geometric distances to p-values, offering a guaranteed confidence [47]. Instead, when the probabilities of a classifier are used alone and the classifier derives its probabilities using any form of scaling (e.g., Platt's scaling used in SVM), the thresholding is more sensitive to outliers.

Function majority voting. Our classification uses functions as the minimum amount of classifiable information. As described in §4.2, these functions are extracted from their parent samples, i.e. larger code samples that authors embed within their posts. It is natural then that most of the functions obtained from the same parent are attributed to the same user. Accordingly, we apply a majority voting algorithm over all the functions that have been extracted from the same parent sample. Once all the classification votes have been counted, the author with most votes is selected as the author of the parent code sample.

This results in greater robustness of the classification, as individual errors do not impact the final classification accuracy, provided that other functions in the same snippet are correctly attributed to the author. This approach improves attribution accuracy, as we later see in §5.2.

Authorship attribution. Our methodology attributes code snippets to accounts based on the function-based majority voting algorithm from the previous step. This approach, by contrast, would not be possible if classification is done based on the entire parent samples (i.e., the snippets). We use as a ground truth label the user account posting the code (either functions or parent samples). This does not necessarily mean that the code is actually written by the person behind this account. In this work, we refer to this *account attribution* as *authorship attribution*, as we consider the posting account to be the author for the purposes of an analyst working with these forums.

5 Results

In this section, we first describe our experimental setting and a set of measures on the GCJ dataset carried out to inform optimal operational settings on the UF dataset. We then present the results of using our methodology, assuming both a closed-world and an open-world.

5.1 Experimental Settings

In this section, we justify some of the experimental settings selected for our evaluation. In particular, we focus on the feature set used for classification, the number of authors included in the experiments, and the minimum size (in LoC) for the samples.

Feature set. Figure 5 shows the performance reached by a code attribution system against GCJ after applying our methodology. The figure shows the breakdown of the classification per type of feature. Syntactical features contribute the most to the performance of the classification, producing results indistinguishable from the entire feature set. This is as a result of the AST node bigrams, which is the feature type which contributes the most to classification. The second most significant feature type is lexical. A similar phenomenon as with the bigrams is also at work here: word unigrams are the second most discriminating feature after AST node bigrams, and produce almost the same accuracy as the rest of the lexical features altogether. Interestingly, as illustrated by Figure 6, with higher numbers of authors lexical word unigrams contribute more to the attribution than the syntactical AST node bigrams. This can be due to the code in the UF dataset having a wide enough range of purposes and contexts such that the classifier relies less on stylistic information and more on the specific content of samples with a higher degree of granularity.

Inspecting the most important features we note differences in the type of features that perform best in the GCJ and UF settings. While 6 of the 10 most important features to the classifier in GCJ settings are AST bigrams, 2 are other syntactical features and one is a lexical feature. When classifying UF code, 4 of the top 10 are lexical and 3 are syntactical, including 1 AST bigram, and 3 features are layout features. For reference, in our experiments, the total number of features in the feature vectors exceeds 400, and there are 6 layout features in the feature set in total, meaning that they are vastly over-represented when classifying UF samples. This can be due to more dissimilar code in UF, making it more easily classifiable by pervasive structural differences rather than subtle stylistic ones.

Number of authors. In both GCJ and UF, we mostly consider an attribution problem with 100 authors. We chose this number since related works on

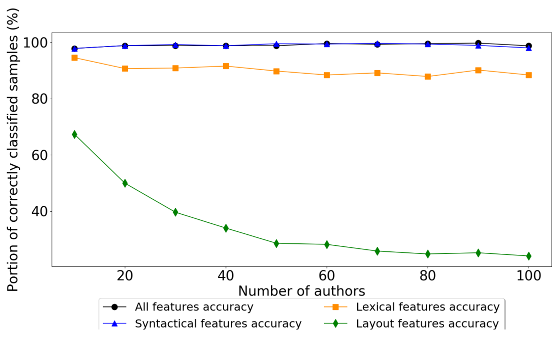

Fig. 5. Relative importance of types of features in attribution tasks varying the number of authors, GCJ.

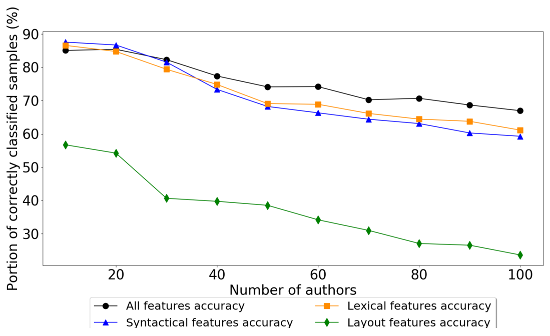

Fig. 6. Relative importance of types of features in attribution tasks varying the number of authors, UF.

code stylometry are also evaluated on samples of 100 authors [1, 14, 16]. We also find some other works looking at 50 authors [4, 34] and 20 authors [2] and we thus also report numbers in these settings. This way, we provide a convenient point of comparison for our experiments. The same results also serve as benchmark numbers to potential underground forums analysts who need to understand what kind of attribution accuracy they can expect under different settings when studying specific smaller subsets of authors (e.g., botnet developers).

Regardless of the number of authors, related works argue that the authorship attribution requires a 'sufficiently large' set of samples from the same author to perform well in difficult conditions [21]. However, it still remains unclear what constitutes a sufficiently large dataset. Moreover, depending on the investigation that needs to be carried out on underground forums, the amount and quality of data available might differ. Thus, we continue our experimentation by analyzing which are the optimal operational settings, and how different settings affect the accuracy of the attribution.

Number of LoC per sample and author. We notice substantial differences between the two datasets (UF and GCJ). We analyze the influence that those differences have in the performance of a classifier in Appendix B. We see that the ideal conditions observed in GCJ, in terms of data available, are not met in the UF dataset. In particular, we study the optimal operational settings that influence the performance of classification with respect to: the number of lines of code (LoC) in the samples, and the number of samples per author. In a nutshell, we see that authorship attribution yields better performance when there is a substantial number of LoC per sample and enough samples per author. However, this improvement plateaus — in GCJ, at $LoC \approx 100$ and with 5 samples per author. We then use this finding to inform our thresholds when dealing with UF in the next section (§5.2). A full breakdown of the experiments around the optimal operational setting can be found in Appendix B.

5.2 Evaluation: Closed-World

We see in §3.2 that the classification performance follows a gradual decline as more authors are considered (Figure 1). In this section, we evaluate our methodology in different settings under a closed-world scenario.

Table 1 shows the effect of each step of our methodology on the accuracy of classification (each row includes the number of the corresponding methodology step shown in Figure 4). As we show next, we evaluate the classification performance in our three datasets: GCJ, GH and UF.

GCJ. Almost every submission in the GCJ dataset consists of only one function. Thus, in practice, the classifier learns on each sample as if we did not have step (1) of our methodology, except those lines outside of a function definition are not included. We see the clone removal procedure (2) on the GCJ dataset removes 157 samples, totaling 0.5% of the sample set. It is possible that the removed functions are copied from internet sources, or they are simply small helper functions which multiple programmers independently write in the same manner. When looking at the per-sample accuracy after using our methodology (see 'Combined steps' in Table 1) in a setting with varying authors (from 20 to 100), we observe similar results than our pre-methodology experiment (see 'Out-of-the-box' row in the same table) albeit there is a slight improvement of 1%.

GH. The accuracies achieved with the GH dataset, although improved by the method, do not approach the accuracies produced in GCJ. There are two major differences between the datasets that might provoke this: firstly, we gather 100 repositories with one contributing author each under the assumption that the code in these repositories has been written by one person. However, given the size and complexity of the repositories, it is likely that at least some of them contain code written by multiple programmers, upsetting the ground truth given to the classifier. Secondly, there is a wider range of code contexts in the repository set than in the constrained set of programming problems in the GCJ contest.

Before function extraction, there are 5,639 samples across 100 authors. Clone removal discards 391 (7%) of the samples. After function extraction, there are 98,028 function samples across the 100 authors. Clone removal discards 33,704 (34%) of these. Again, much of the code re-use appears on the function-level as opposed to entire source files being copied. Even when removing over a third of the dataset however, in GH the accuracy does not significantly improve.

Underground forums. Our methodology has a significant positive effect on authorship attribution accuracy in UF. In fact, we see that each step results in an improvement. When performing attribution on 100 authors we double the accuracy, with the difference in attribution accuracy with and without following our methodology being 44%. With fewer numbers of authors, the improvement is slightly less pronounced, al-

	Per-sample accuracy								
Step of methodology applied	**UF 20**	**GCJ 20**	**GH 20**	**UF 50**	**GCJ 50**	**GH 50**	**UF 100**	**GCJ 100**	**GH 100**
Out-of-the-box	66%	99%	81%	48%	99%	63%	33%	98%	55%
(2) Clones removed	66%	99%	81%	50%	99%	65%	34%	98%	55%
(3) Sample pruning	68%	100%	85%	53%	98%	64%	44%	99%	60%*
(4) Confidence thresholding	78%	100%	81%	69%	100%	65%	47%	100%	68%
(5) Split into functions: majority voting	84%	97%	94%	79%	97%	71%	60%	97%	71%
Combined steps (1, 2, 3, 4, 5)	89%	100%	94%	82%	100%	75%	77%	99%	71%*

Table 1. Effect on attribution accuracy of steps in the methodology on Google Code Jam and Underground Forums code in an author closed-world attribution task, varying the number of authors. *These experiments are done with 86 authors due to sample pruning.

though the overall performance is better: when performing attribution on 20 authors our method improves the accuracy by 23% (from 66% to 89%).

There are 18,853 clones among complete snippets, totaling 20.9% of the total dataset. Splitting the code samples into constituent functions (1) produces 39,661 function samples of which 18.6% are removed after employing the clone removal method at the function level (2). This shows, first, that a significant number of code snippets posted by users do not contain any identifiable function definitions at all, as 39,661 functions are extracted from 71,232 parent samples. Second, it shows that code re-use often happens on a function level where a user posts a code snippet which can be said to be their own but re-uses other users' functions. This can be for a number of reasons, e.g. each user offering their own solution to a problem presented in a thread, leading to code commonalities alongside distinct code.

While splitting the code into functions results in significant accuracy improvements, the lack of orphan lines as a result of this could have a deleterious effect due to further limiting the size of the samples the classifier is able to learn on. In our experimentation with 100 authors this did not affect accuracy. However, it resulted in a 7% drop with 50 authors and a 5% drop with 20 authors. Code re-use in this domain is rampant, and although a significant amount of code clones is removed, this step has a minor effect on the accuracy of attribution. Only a minority of low-interaction users post code clones (see Appendix C), thus the classifier is able to overcome the noise introduced by these samples on a per-author basis.

Maximizing the number of LoC per sample (3) grants a considerable boost to accuracy, at 100 authors from 33% to 44%. This is when discarding samples under 25 LoC, which is the minimum required to consider a 100-author dataset with at least 5 samples per author (analysis details are in Appendix B). This combined with the finding that optimal accuracy in a curated

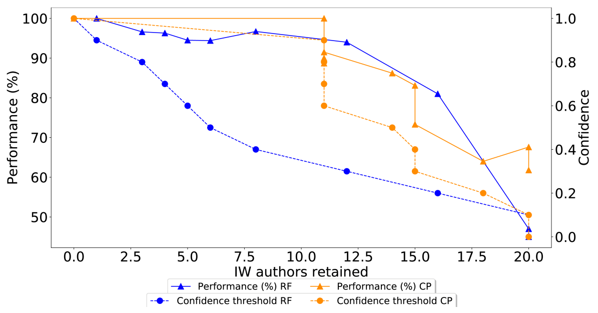

Fig. 7. Thresholding samples based on confidence, per-sample accuracy and percentage of authors retained. Confidence is calculated by Random Forest classifier probability (RF) or Conformal Predictor confidence (CP). Open-world task with 20 in-world and 80 out-of-the-world authors.

dataset is only achieved at 100 LoC suggests that the scarcity of stylistic data in samples due to their brevity is a significant limiting factor to successfully apply code stylometry in underground forums. This effect increases in strength with the number of authors used for classification, and is less pronounced in the 20 author task.

Finally, thresholding samples based on Conformal Predictor confidence (4) allows us to discard samples which have a high probability of being wrong due to low classifier certainty. Our approach is in this way ambivalent to the open-world assumption (see §5.3), as out-of-world samples tend to be predicted with low confidence. For our purposes, we use a threshold of 50% confidence. Under this condition, one-third of the predictions are rejected, and increasing this threshold in our UF domain leads to diminishing returns. We manually validate 10% of classifications to the best of our ability and verify the veracity of our ground truth labels.

5.3 Evaluation: Open-World

The closest work to ours [21] addresses the open-world question by setting a threshold of classifier confidence under which they reject the classification. The assumption is that low confidence implies that a sample might not have been written by an author in the known suspect set (i.e. it is from an out-of-world author). There is an important side effect when rejecting samples based on a confidence threshold: in-world samples may be rejected to a point where an author can be left out.

We compare our methodology by judging the performance with respect to the number of in-world authors that are discarded at different confidence thresholds. Thus, we construct a set of calibration curves and use a probability-based rejection method as in [21]. We then assess how CP performs at rejecting authors — recall that CP rejects based on p-values.

Figure 7 shows this comparison where the RF setting represents the results of calculating confidence using probabilities in the UF dataset. We retain 20 authors as in-world and 80 as out-of-world, meaning approximately 80% of samples in these experiments are out-of-world. In the out-of-the-box setting (i.e., without applying our methodology's steps), and without rejecting any samples, we achieve an accuracy of 45% (see the rightmost side of the graph), dropping the accuracy from 66% in the closed-world task (see Table 1 in §5.2). This accuracy recovers when we use 20% classifier probability confidence as our sample acceptance threshold, and improves to 97% when rejecting samples below 40% confidence. In our domain, this method results in a significant trade-off having to be made. Even at the second-lowest acceptance threshold of 20% confidence, while we discard 48% of out-of-world samples, we also discard over 22% of in-world samples. At the point where almost every out-of-world sample is filtered out (confidence of 40% — 94% of out-of-world samples discarded), the majority (62%) of in-world samples are also discarded. Thus, this method is not appropriate for delineating between in-world and out-of-world data points, however, the calibration curves can aid the security analysts and law enforcement officers to set expectations of how certain they can be in the predictions provided by the classifier in this domain [21].

Our own approach using a Conformal Predictor (CP), instead of Random Forest classifier (RF) probabilities, achieves similar performance overall at a much lower cost in terms of in-world samples rejected. In particular, we see that CP yields higher confidence at all points in the X-axis while retaining a considerably larger number of in-world samples than RF probabilities. For instance, at a RF probability threshold of 40%, 94% of out-of-world samples are rejected but only 6 in-world authors are retained. Using confidence provided by the CP process, at a threshold of 90%, 93% of out-of-world samples are rejected while it retains 15 (out of 20) in-world authors. This altogether makes the use of the CP method for calculating confidence more tolerant than using RF probabilities — using RF classifier probability consistently retains fewer authors than using CP confidence. As a result, under most conditions using the CP confidence provides a favorable trade-off. We also see with CP interesting nuanced trade-offs, for instance, we can increase the confidence threshold from 0.6 to 0.9 without rejecting any additional in-world authors and boosting the accuracy from 80% to 99%.

6 Case Study: Multi-Accounts

We use our methodology to attempt to identify accounts belonging to the same user in different forums (inter analysis) as well as on the same forum (intra analysis).

6.1 Inter-Forum Analysis

We use the largest forum to build a model (it includes 21,359 function samples). We use this model to make predictions on the second largest, yielding a set of 9,736 function samples over 2,515 user accounts. As we do not have a ground truth of multi-accounts (namely doppelgänger) across forums, we assess the potential application of code stylometry to link together accounts that belong to the same programmer through a case study. In particular, we next report two pairs of accounts which have their samples consistently classified as their counterpart. We are confident that these two accounts belong to the same author because the usernames used in both forums are identical, but also in light of the manual analysis we perform. Most of the samples classified in the following case studies are classified with a confidence of 85%+ as provided by the Conformal Predictor.

Case Study I. This user has 46 function samples in the training forum and 7 samples in the target forum. We successfully classify 6 of these functions as theirs. We manually verified this samples as snippets of hacks for a specific video game (i.e., Minecraft).

This is a particularly interesting case. In one of the forums, the user released various hacks and cheats related to two popular video games, i.e., Call of Duty and Minecraft. Meanwhile, in the other forum, the user was active in generic discussions about video games and also other miscellaneous topics (i.e., graphics, music or video editing). At a certain date, the activity in the gaming forum stopped, and at the same time the actor changed its username in the other forum. In this case, due to historical view of the CrimeBB dataset [36], we were able to observe that previous posts were made with the former username, which enabled us to link these two accounts. Still, in absence of such a historical view, our methodology can successfully identify the accounts as being from the same user.

Case Study II. This user has 346 function samples in the training forum, but only 10 function samples in the target forum. Also, their test samples are very short, usually 1-3 LOC, making attribution exceptionally difficult. Still, our method classifies 2 of these functions as theirs. We observe that this user used one of the forums (more hacking-oriented) for questions and troubleshooting related to C/C++ programming, while the other forum (more game-cheating oriented) was used for the trading of bots and cheats related to one popular video-game. Interestingly, the first activity in both forums was related to the request for advice on how to code exploits and tutorials on hacking topics, and the trading of virtual game items respectively. After various years on the two forums, the actor released hacking material (e.g., spamming tools, SMS bomber or social media hacking). This analysis confirms previous observations that indicated a common path followed by these actors, from gaming-related topics, to cybercrime activities [24, 33]. It also shows the evolution of the expertise acquired on these forums [35].

Consistent classifications on other usernames. We use the two previous case studies to inform our confidence thresholds for finding potential multi-accounts across forums. The confidence score provided by the Conformal Predictor tends to be high for these samples (i.e., 85% to 95%) with some exceptions which have low confidence (30%). Using 85% confidence as a threshold, we observe that there are 16 pairs of accounts, including the two mentioned above in this subsection, with two or more snippets classified. Each pair consists of a training partner from one forum (A) which provided samples S from a username u (S_u) to train the model, and a testing partner v from another forum (B, where $B \neq A$) whose samples (S_v) were classified as belonging to the training partner ($\texttt{predict}(S_u) = v$). While we perform a manual validation of the 16 pairs as we discuss next, we note that drawing conclusions without a ground truth or clear signals such as the ones found in the two case studies above is a matter of epistemic uncertainty.

While we see in all of the 16 pairs predictions with strong confidence, we also see a significant number of contradictions (i.e., some of testing partner's samples not being predicted as belonging to the training partner while others are). This makes the multi-account hypothesis weaker for most of the pairs. However, there is a pair where the testing partner has most of their samples (31 out of 32) classified to its training partner, with 2 of them passing the 85% confidence threshold. Moreover, the training partner has only 10 samples, theoretically covering a small stylistic range. However, this same training partner is also present in predictions from 5 of the 16 pairs, which could suggest several things: 1) that their style lends itself well to being misclassified, 2) that a number of authors re-posted others' samples, or 3) that the very same actor has more than one account in the testing forum.

We also see other contradictions that might stem from the way the dataset is split. Note that the training and the testing set in an inter-forum setting can be conspicuously imbalanced as some forums are significantly more popular than others. For instance, one training account appears in 6 of the 16 pairs, having contributed 175 samples to the training set. In another pair, the testing partner only has two samples, both of which are classified as the same training partner with over 85% confidence, the training partner having contributed only with 10 samples to classifier training.

There are 815 user accounts in the testing set which carry at least 3 function samples. Of these, around 300 (37%) have at least 70% of their samples classified consistently as a specific user account from the training set. While this gives the analyst a starting point for trying to detect accounts across the two forums belonging to the same author, it is clear to see that consistent classification alone is not a reliable metric for this purpose.

6.2 Intra-Forum Analysis

We study the existence of within-forum multi-accounts in the top 100 accounts used for our other experiments explained in §5. We look at pairs of accounts where most of the samples in one of the accounts are confidently classified as belonging to the other account (according to a Conformal Predictor confidence of 85% and over). We identify 5 such pairs, however upon manual investigation (i.e., the analysis of the code snippets and the text posted by these users) we can not conclusively ascertain whether the accounts indeed belong to the same actor, highlighting the difficulty of this task when performed by a human analyst.

7 Discussion

In this section, we first discuss the limitations of the methodology and then our main take-aways.

7.1 Limitations

While our methodology outperforms current the state of the art techniques, our results are still significantly worse than those obtained in laboratory conditions (i.e., on datasets such as GCJ). This is due to various factors.

First, the presence of *code-reuse*. While we remove code clones from the dataset, these do not account for cases where, for example, one author copies from another's code and partially modifies it, without adding substantial stylistic changes that tell these two apart.

This limitation could be overcome using more exhaustive code plagiarism detection techniques. In our own experiments, we explored different metrics of code similarity in order to find code clones as well as a range of thresholds for flagging a sample as a clone. The threshold we settled at was found to produce the best results, although we leave using token-based and sequence-based techniques to future work.

Second, during our experimentation, we have deconstructed the GCJ dataset to analyze which are the optimal *operational settings* for doing attribution. Thus, in some steps we have discarded samples that are more likely to be misattributed (e.g., due to lack of enough LoCs). Depending on the specific scenarios, this might not be an option for the analyst. However, our main goal is to flag existing limitations in the state of the art, showing which factors affect attribution accuracy. Our methodology can help analysts to contextualize and interpret better attribution results.

Third, the lack of enough *sampling data*. We have observed that, besides the authenticity of the data, the amount of such data is paramount for proper attribution. In GCJ, the ideal number of LoC per sample for attribution is 100. However, this refers to a setting where such an amount of data might be available. In the case of underground forums dataset, if we set a minimum number of 100 authors to analyze, the analyst is left with samples over 25 LoC. Still, using this threshold results in a considerable accuracy improvement (from 33% to 44%) with respect to the out-of-the-box application of the state of the art on underground forum samples (see Table 1). When considering the top 10 authors, in the same way, we are able to gather only samples of at least 45 LoC. Only a small minority of code snippets have 100 LoC (after function splitting). This implies that, if the dataset had enough samples per author for every author to possess 100-line samples, the accuracy might be in line with the attribution accuracy achieved with ideal datasets (GCJ).

Fourth, while the domain is by nature adversarial, we assume that authors do not take steps to obfuscate their programming style. Indeed, there are attempts to hinder authorship attribution on natural language to enhance author privacy [41]. We leave the evaluation of the effects of such techniques on our method and their applicability to source code to future work.

Finally, our work inherits limitations from previous works addressing authorship attribution in underground forums. *Doppelgänger finder* is a stylometry analysis tool [4] which suffers to deal with 50 authors (for performance issues), and requires documents of at least 10K words (for optimal accuracy) [34]. Again, these constraints might limit the use of these tools, but can still be useful to assist analysts.

7.2 Take-Aways

To the best of our knowledge, we are the first to address the code stylometry problem using real data from underground forums. Code authorship attribution in these scenarios poses additional challenges and existing out-of-the-box solutions do not work well (see §3).

Effect of plagiarism. We observe that removing exact code clones has a negligible effect on attribution accuracy compared to the other steps (see Table 1). To confirm this, we test the method on an additional dataset of source code extracted from post attachments in a subset of the CrimeBB dataset,[4] and obtained similar conclusions. There were 36k samples yielded by these attachments, of which 49% were identified as identical code clones and removed. Before clone removal, attribution accuracy was 16%, whereas afterward, it increased to 28%. Code clones then appear to have a lesser effect on accuracy than one would expect: even when removing half of the samples which are all clones, accuracy

4 This happens when users share more substantial amounts of code, such as entire projects.

improved only by 12%. Likewise in the UF samples, removing 20.9% of them increased accuracy by 1%, and in GH removing 34.4% of function samples only increases accuracy in the 50 author setting and only by 2%.

The importance of the context. Additionally, while the code in the Google Code Jam contest solves a well-defined set of different programming problems, the purposes of code snippets posted on underground forums are unknown (e.g., video-game hacks, malware, or other offensive tools). This contributes to a negative effect on the performance of the attribution task, as a classifier can adapt to the type of problem being solved rather than the coding style.

Function-level granularity. Overall, our results show that our proposed methodology aids in the development of a machine learning pipeline that has constraints that stem from the data collection. We show that the granularity of the feature extraction process is paramount in difficult situations. When programmers reuse code, they generally adapt it to their context, introducing their own programming style and mixing it with the stylistic features already present, to varying degrees. In this regard, splitting snippets into functions is beneficial because the attribution is done over well-defined structures (even at the cost of removing orphan lines). Thus, we introduce a voting system to classify each code snippet based on the different functions it contains.

Applications. Our methodology makes a step-forward on the code stylometry analysis in underground forums. While it further exposes a potential threat to the privacy of the actors, who typically act in these forums with a sense of anonymity, it might be of potential interest for law enforcement agencies or security practitioners to assist during online investigations. It helps to attribute samples to accounts, regardless of whether the account has multiple authors behind it. In §6 we showcase examples of different accounts belonging to the same user being detected by means of their programming style. We also show how stylometry survives over the years and across different coding activities like game hacking, spamming or social media hacking, and discuss implications of our work to understand the different pathways to crime. We believe that our methodology complements other existing efforts for online account attribution [4, 38, 52]. The use of these in combination is left for future work. Also, our experimental data is useful to analysts to understand the accuracy they can expect when applying the methodology to more broad settings such as 100 authors, as well as smaller sets of authors when wanting to de-anonymize a narrower subset of users (e.g., botnet developers on a particular forum).

Classifier confidence. Performing confidence thresholding is a powerful technique which prevents the analyst from having to sift through predictions that are little better than guesses, as well as improves accuracy in the predicted samples. In tasks using the open-world assumption, this method helps to filter out samples with authorship outside the suspect set. However, it does not provide a complete solution to delineating between in-world and out-of-world samples. This is still an open problem [21], although our system performs rejection more selectively and accurately than related work. In the closed-world setting it is one of the methods used to counter-act the uncertainty of ground truth inherent in data gathered from underground forums. Using it in conjunction with other methods aimed at dealing with noisy labeling in machine learning [12, 39, 53] is precisely the scope of our future work.

Programming language. We note that our methodology is agnostic of the programming language and that the expressiveness may not be noticeable across languages as our method uses Abstract Syntax Trees that decouple our system from traits unique to the language. We select C/C++ since it is the most representative programming language in our dataset, and indeed, it is the defacto programming language for hacks (e.g., exploits or malware) and video game cheats. Thus, developing stylometry for C/C++ is of particular relevance in the context of underground activities. However we discard 84.5% of samples due to filtering for C and C++ (see Appendix A). Programmers may unconsciously tailor their style to the programming language they use and this leaves open the question of how the results will vary, if at all, when applied to other programming languages. Our method can be applied to any language as long as an Abstract Syntax Tree is able to be generated. Thus, we posit that our method should be able to capture style changes across programming languages for the same developer if we have sufficient number of samples. Evaluating this is the scope of our future work.

Classification algorithms. In our methodology, we have applied Random Forests due to its simplicity and probed accuracy to deal with the code stylometry problem [16]. A potential area of research is to apply advanced techniques from Deep Learning to address this problem [7]. Indeed, recent advances for image classification using Convolutional Neural Networks (CNNs) successfully deal with noisy labels [26]. Also, CNNs can be improved to deal with out-of-distribution detection in production time [19]. While these improvements are tailored to the problem of image classification, similar techniques could be adopted for the source code attribution problem with noisy or unreliable labels, as in the case of underground forums. A potential challenge to address, however, is the need to train DL models with datasets with low number of samples per author. Also, this is an active area of research, and it is known that criminals adapt their techniques to cheat the system [25]. Thus, we believe that future research efforts should focus on the application of robust techniques like DL, as well as their application in adversarial settings [37].

8 Related Work

Underground forum analysis. While related work performs authorship attribution in underground forums using different signals, such as the style [4, 38, 44] or the media content posted [52], to the best of our knowledge, this is the first work dealing with source code directly.
Code Stylometry. Caliskan-Islam et al. [16] introduced the usage of random forest to perform authorship attribution from source code samples, to which end they introduced the Code Stylometry Feature Set. Authors in [21] used the CSFS to perform authorship attribution in software repositories. They were not able to match results from [16] obtaining between 50% and 60% accuracy. Other related works such as [6, 45, 51] have proposed alternative machine learning algorithms, but again test their methods on either the GCJ dataset or in the case of [6], scraped GitHub code not subject to many of the constraints of our domain.

There exist alternative algorithms which have been used with natural language which are promising directions for research in code stylometry [9, 42], the exploration of which is out of scope for this work.
Out of the lab. While a few previous works on authorship attribution achieve competitive results in difficult conditions [6, 21], these methods require a sufficiently large set of samples belonging to the same author. This is not generally the case in underground forums as code posts tend to have incomplete code fragments and not all users are equally prolific at sharing source code.
Learning with confidence. Dauber et al. [21] construct calibration curves in order to provide to the analyst information with which to set thresholds to reject low classification confidence samples. While useful, they use the classifier's output probabilities as the classifier's confidence in its predictions. In our work, we employ a more advanced statistical tool, a Conformal Predictor method [47], in order to obtain more reliable confidence scores for samples. Conformal evaluation has been used to aid in solving the malware classification problem [10, 20, 28, 29]. To the best of our knowledge, we are the first to show that Conformal Prediction theory can be used to improve code stylometry in settings where obtaining a clean and curated dataset is challenging.

9 Conclusions

In this work, we study the key factors influencing the performance of existing code stylometry techniques, as well as a set of strategies to mitigate issues in difficult conditions. We propose a methodology that increases the accuracy of state-of-the-art techniques by 44% when performing attribution in underground forums while performing at the same level when dealing with curated datasets already used in previous works. For a scenario with 100 authors, our methodology more than doubles the performance of the classification task, while achieving 89% accuracy with a less demanding configuration of 20 possible authors. Our work raises important open questions about these factors, mainly around the prominence of code re-use and its implications in authorship attribution. Our methodology provides additional insights to assist in the identification of users posting across different forums, even when the classifier has been trained with a single one. This is possible thanks to the usage of CP to discard those matches with very low confidence. We also provide insights into the type and qualities of code one can find in underground forums and how these impact an analyst's effort to perform authorship attribution. Our work carries implications of how to avoid detection as a user of these forums: keep snippets of code as short as possible, copy others' code (even if *dead code*), where possible avoid submitting code that can be easily parsed, and if wanting to post multiple functions, split these up into multiple posts so as to counteract the functions voting for the right author for classification.

10 Acknowledgements

This work has been partially funded by the Spanish grants ODIO (PID2019-111429RB-C21 and PID2019-111429RB) and the Region of Madrid grant CYNAMON-CM (P2018/TCS-4566), co-financed by European Structural Funds ESF and FEDER, the Excellence Program EPUC3M17, and the "Ramon y Cajal" Fellowship RYC-2020-029401.

References

[1] Abbasi, A. and Chen, H. (2006). Visualizing authorship for identification. In *International Conference on Intelligence and Security Informatics*, pages 60–71. Springer.

[2] Abuhamad, M., AbuHmed, T., Mohaisen, A., and Nyang, D. (2018). Large-scale and language-oblivious code authorship identification. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 101–114.

[3] Afroz, S. (2020). How to build realistic machine learning systems for security? San Francisco, CA. USENIX Association.

[4] Afroz, S., Islam, A. C., Stolerman, A., Greenstadt, R., and McCoy, D. (2014). Doppelgänger finder: Taking stylometry to the underground. In *2014 IEEE Symposium on Security and Privacy*, pages 212–226. IEEE.

[5] Allodi, L. (2017). Economic factors of vulnerability trade and exploitation. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1483–1499.

[6] Alsulami, B., Dauber, E., Harang, R., Mancoridis, S., and Greenstadt, R. (2017a). Source code authorship attribution using long short-term memory based networks. In *European Symposium on Research in Computer Security*, pages 65–82. Springer.

[7] Alsulami, B., Dauber, E., Harang, R., Mancoridis, S., and Greenstadt, R. (2017b). Source code authorship attribution using long short-term memory based networks. In *European Symposium on Research in Computer Security*, pages 65–82. Springer.

[8] Anderson, R., Barton, C., Bölme, R., Clayton, R., Ganán, C., Grasso, T., Levi, M., Moore, T., and Vasek, M. (2019). Measuring the changing cost of cybercrime.

[9] Bagnall, D. (2016). Authorship clustering using multi-headed recurrent neural networks. *arXiv preprint arXiv:1608.04485*.

[10] Barbero, F., Pendlebury, F., Pierazzi, F., and Cavallaro, L. (2020). Transcending transcend: Revisiting malware classification with conformal evaluation. *arXiv preprint arXiv:2010.03856*.

[11] Bhalerao, R., Aliapoulios, M., Shumailov, I., Afroz, S., and McCoy, D. (2019). Mapping the underground: Supervised discovery of cybercrime supply chains. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–16. IEEE.

[12] Biggio, B., Nelson, B., and Laskov, P. (2011). Support vector machines under adversarial label noise. In *Asian conference on machine learning*, pages 97–112. PMLR.

[13] Bogomolov, E., Kovalenko, V., Bacchelli, A., and Bryksin, T. (2020). Authorship attribution of source code: A language-agnostic approach and applicability in software engineering. *arXiv preprint arXiv:2001.11593*.

[14] Burrows, S. and Tahaghoghi, S. M. (2007). Source code authorship attribution using n-grams. In *Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University*, pages 32–39. Citeseer.

[15] Caliskan, A., Yamaguchi, F., Dauber, E., Harang, R., Rieck, K., Greenstadt, R., and Narayanan, A. (2015). When coding style survives compilation: De-anonymizing programmers from executable binaries. *arXiv preprint arXiv:1512.08546*.

[16] Caliskan-Islam, A., Harang, R., Liu, A., Narayanan, A., Voss, C., Yamaguchi, F., and Greenstadt, R. (2015). De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security), Washington, DC*.

[17] Calleja, A., Tapiador, J., and Caballero, J. (2016). A look into 30 years of malware development from a software metrics perspective. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 325–345. Springer.

[18] Ceschin, F., Gomes, H. M., Botacin, M., Bifet, A., Pfahringer, B., Oliveira, L. S., and Grégio, A. (2020). Machine learning (in) security: A stream of problems. *arXiv preprint arXiv:2010.16045*.

[19] Chen, J., Li, Y., Wu, X., Liang, Y., and Jha, S. (2020). Robust out-of-distribution detection for neural networks. *arXiv preprint arXiv:2003.09711*.

[20] Dash, S. K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., and Cavallaro, L. (2016). Droidscribe: Classifying android malware based on runtime behavior. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 252–261. IEEE.

[21] Dauber, E., Caliskan, A., Harang, R., Shearer, G., Weisman, M., Nelson, F., and Greenstadt, R. (2019). Git blame who?: Stylistic authorship attribution of small, incomplete source code fragments. *Proceedings on Privacy Enhancing Technologies*, 2019(3):389–408.

[22] Dong, W., Feng, Z., Wei, H., and Luo, H. (2020). A novel code stylometry-based code clone detection strategy. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1516–1521. IEEE.

[23] Google (2008). Google code jam. https://web.archive.org/web/20080830055526/https://code.google.com/codejam.

[24] Hughes, J., Collier, B., and Hutchings, A. (2019). From playing games to committing crimes: A multi-technique approach to predicting key actors on an online gaming forum. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12. IEEE.

[25] Hutchings, A., Pastrana, S., and Clayton, R. (2019). Displacing big data: How criminals cheat the system. *Cybercrime: The human factor. Oxon, UK: Routledge*.

[26] Jiang, L., Huang, D., Liu, M., and Yang, W. (2020). Beyond synthetic noise: Deep learning on controlled noisy labels. In *International Conference on Machine Learning*, pages 4804–4815. PMLR.

[27] Joern (2019). Joern. https://joern.io/.

[28] Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., and Cavallaro, L. (2017). Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642.

[29] Jordaney, R., Wang, Z., Papini, D., Nouretdinov, I., and Cavallaro, L. (2016). Misleading metrics: On evaluating machine learning for malware with confidence. *Tech. Rep.*

[30] Kantchelian, A., Tschantz, M. C., Afroz, S., Miller, B., Shankar, V., Bachwani, R., Joseph, A. D., and Tygar, J. D. (2015). Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 45–56.

[31] Krebs, B. (2017). Who is Marcus Hutchins?

[32] Motoyama, M., McCoy, D., Levchenko, K., Savage, S., and Voelker, G. M. (2011). An analysis of underground forums. In *Proceedings of the 2011 ACM SIGCOMM conference on*

Internet measurement conference, pages 71–80.

[33] National Crime Agency (2017). Pathways into cyber crime.

[34] Overdorf, R. and Greenstadt, R. (2016). Blogs, twitter feeds, and reddit comments: Cross-domain authorship attribution. *Proceedings on Privacy Enhancing Technologies*, 2016(3):155–171.

[35] Pastrana, S., Hutchings, A., Caines, A., and Buttery, P. (2018a). Characterizing eve: Analysing cybercrime actors in a large underground forum. In *International symposium on research in attacks, intrusions, and defenses*, pages 207–227. Springer.

[36] Pastrana, S., Thomas, D. R., Hutchings, A., and Clayton, R. (2018b). Crimebb: Enabling cybercrime research on underground forums at scale. In *Proceedings of the 2018 World Wide Web Conference*, pages 1845–1854.

[37] Quiring, E., Maier, A., and Rieck, K. (2019). Misleading authorship attribution of source code using adversarial learning. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 479–496.

[38] Rocha, A., Scheirer, W. J., Forstall, C. W., Cavalcante, T., Theophilo, A., Shen, B., Carvalho, A. R., and Stamatatos, E. (2016). Authorship attribution for social media forensics. *IEEE Transactions on Information Forensics and Security*, 12(1):5–33.

[39] Sabzevari, M., Martínez-Muñoz, G., and Suárez, A. (2018). A two-stage ensemble method for the detection of class-label noise. *Neurocomputing*, 275:2374–2383.

[40] Samtani, S., Chinn, R., and Chen, H. (2015). Exploring hacker assets in underground forums. In *2015 IEEE international conference on intelligence and security informatics (ISI)*, pages 31–36. IEEE.

[41] Shetty, R., Schiele, B., and Fritz, M. (2018). A4nt: author attribute anonymity by adversarial training of neural machine translation. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1633–1650.

[42] Shrestha, P., Sierra, S., González, F. A., Montes, M., Rosso, P., and Solorio, T. (2017). Convolutional neural networks for authorship attribution of short texts. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 669–674.

[43] Soska, K. and Christin, N. (2015). Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *USENIX Security Symposium*, pages 33–48.

[44] Spitters, M., Klaver, F., Koot, G., and van Staalduinen, M. (2015). Authorship analysis on dark marketplace forums. In *2015 European Intelligence and Security Informatics Conference*, pages 1–8. IEEE.

[45] Ullah, F., Wang, J., Jabbar, S., Al-Turjman, F., and Alazab, M. (2019). Source code authorship attribution using hybrid approach of program dependence graph and deep learning model. *IEEE Access*, 7:141987–141999.

[46] Van Wegberg, R., Tajalizadehkhoob, S., Soska, K., Akyazi, U., Ganan, C. H., Klievink, B., Christin, N., and Van Eeten, M. (2018). Plug and prey? measuring the commoditization of cybercrime via online anonymous markets. In *27th USENIX security symposium (USENIX security 18)*, pages 1009–1026.

[47] Vovk, V., Gammerman, A., and Shafer, G. (2005). *Algorithmic learning in a random world*. Springer Science & Business Media.

[48] Vu, A. V., Hughes, J., Pete, I., Collier, B., Chua, Y. T., Shumailov, I., and Hutchings, A. (2020). Turning up the dial: the evolution of a cybercrime market through set-up, stable, and covid-19 eras. In *Proceedings of the ACM Internet Measurement Conference*, pages 551–566.

[49] Wang, N., Ji, S., and Wang, T. (2018). Integration of static and dynamic code stylometry analysis for programmer de-anonymization. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, pages 74–84.

[50] yoeo (2020). Guesslang. https://github.com/yoeo/guesslang.

[51] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., and Liu, X. (2019a). A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 783–794. IEEE.

[52] Zhang, Y., Fan, Y., Song, W., Hou, S., Ye, Y., Li, X., Zhao, L., Shi, C., Wang, J., and Xiong, Q. (2019b). Your style your identity: Leveraging writing and photography styles for drug trafficker identification in darknet markets over attributed heterogeneous information network. In *The World Wide Web Conference*, pages 3448–3454. ACM.

[53] Zhou, X., Ding, P. L. K., and Li, B. (2019). Improving robustness of random forest under label noise. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 950–958. IEEE.

Appendix

A Characterization & Comparison

We have seen that existing stylometry techniques perform worse with UF than with GCJ. In this section, we characterize and compare these datasets, and offer statistics on the GH dataset.

A.1 Types of Posts

The UF dataset consists of 580,432 scraped posts from 53,820 user accounts in 7 different underground forums. We focus our analysis on posts that have code snippets, and in particular, we retain C/C++ code fragments. Thus, we study 90,085 code snippets posted from 19,845 accounts. Out of all 7 forums, 2 make up to 99.5% of the code samples. One is a popular beginner-friendly forum that has large sections dedicated to teaching both hacking techniques and coding in general. The other forum is specialized in video game hacks and cheats, with dedicated sections for coding and programming of these.

A.2 Length and Number of Samples

Table 2 compares the lengths of code snippets for the three datasets. Overall, we observe that GH samples are larger than those from GCJ and UF. This is a result of the repositories containing full software projects as opposed to entries to a coding contest or forum posts. Regarding the others, we observe that samples in GCJ are larger than those from UF. In particular, the average number of lines in UF is 20.9, whereas in GCJ is 74.3. Authors in GCJ tend to have a slightly larger number of samples, although the standard deviation is higher in UF. This means that GCJ is more balanced in terms of the amount of information provided by each class (author). The amount of code written per author is considerably higher in GCJ with 493.4 lines per author and only 92.2 lines in UF. The nature of the tasks achieved by the code in GCJ requires their authors to write more lines. In fact, when looking at the median, GCJ is close to the average with 451 lines per author but UF has only 9 lines per author as median. This means that UF contains many more samples that are considerably shorter than GCJ. The shorter samples in UF are aligned with the kind of usage that users make of these forums, i.e. as a way to exchange knowledge and to increase their reputation within the community [33, 36].

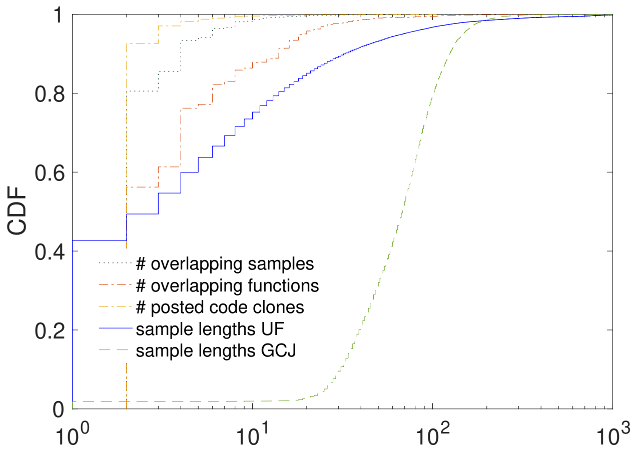

Fig. 8. CDFs of number of snippets functions and samples overlapping with another account (from both UF and GCJ). The graph also shows the number of posts for a given clone (*# posted code clones*). The Y-axis represents the frequency and the X-axis represents the index of the most frequent snippets/functions. The X-axis is truncated to 2000 samples as the remaining have all a frequency of 2.

A.3 Code Re-Use in Underground Forums

We look at code re-use from two different angles: entire snippets and functions. We consider two samples as clones whenever their Hamming distance is less than 1%. Figure 8 summarizes the code reuse for these two units of information in UF.

Snippets. When treating each entire code snippet posted as one sample, we detect clones from 41.2% of the accounts. Figure 8 shows the maximum number of mutual samples any one account has with any other account (see *# overlapping samples*). We see that the variation is high, from 2 to over 100 mutual samples on both ends of the CDF.

Altogether, we observe 18,853 exact code clones, out of which 13,162 (69.8%) are only duplicated once. The distribution can be seen by looking at *# posted code clones* in the same figure. In total, 32% of code clones appear more than twice, with a large proportion of samples being cloned frequently (the highest incidence rate reaching 109 times).

Functions. Functions are only successfully extracted from 26.8% of accounts that have posted C/C++ code snippets. This suggests that most accounts do not post snippets containing function definitions, but only orphan lines;[5] this is corroborated by Figure 8 which shows that in fact, the majority of samples are of length below 10 lines. Oftentimes an account will post a snippet of code which is intended as a correction for code already present in the thread, which means the code posted will be incomplete and not concern itself with properly encapsulating code blocks in functions. Overall, we extract 39,661 functions, with 7,403 (18.7%) being code clones. Function clones are slightly less popular than snippet clones (incidence of 20.9%).

The function-wise distribution in Figure 8 corresponds closely with the distribution of account overlap of parent samples.[6] All non-unique functions extracted from authors' samples originate from a subset of 1,585 accounts.

It is worth mentioning that functions were only successfully extracted for 5,332 accounts, and thus the proportion of these authors with non-unique functions is 29.7%. This shows that the majority of accounts post

5 An **orphan line** is a line of code from a snippet which is not part of a function.

6 In this work, we call a **parent sample** of a function to the original snippet posted from which the function is extracted.

	Samples/author			LoC/author			LoC/sample		
Measure	**GCJ**	**UF**	**GH**	**GCJ**	**UF**	**GH**	**GCJ**	**UF**	**GH**
Mean	6.6	4.5	56.4	493.4	92.2	30,415.6	74.3	20.9	539.4
Median	6	2	16	451	9	4421	68	3	158
Std	1.2	16.5	119.1	266.4	543.0	68,837.1	46.2	105.1	4,047.0

Table 2. Mean, median and standard deviation of samples and LoC per author and LoC per sample in the UF, GCJ and GH datasets.

unique functions, and code re-use on the function level is concentrated in 8.0% of all accounts only.

Interestingly, the standard deviation of non-clone function lengths is 43.79 while the standard deviation of duplicate function lengths is 20, implying that non-unique functions are more consistent in their lengths, where non-clones are distributed with more variance. Clones are also slightly shorter, with a mean of 14.6 lines long versus unique functions' 17.2.

By manual analysis, we observe that the most common functions tend to be written for highly context-dependent purposes as opposed to generic functions written to, for example, return the hexadecimal value of a decimal number (see Appendix 6 for a case study).

B Optimal Operational Setting

In Appendix A we report substantial differences between the two UF and GCJ datasets. Here, we analyze how such differences affect the accuracy of the classifier. As mentioned earlier, ground truth in the GCJ dataset is reliable. Thus we assess on this dataset the optimal operational settings that influence the performance of classification with respect to two main differences: the number of lines of code (LoC) in the samples, and the number of samples per author.

Number of LoC per sample. We first analyze the effect of the total number of LoC on the accuracy when using the GCJ for a 100-author attribution task. We truncate the samples to certain lengths. In particular, we consider samples of variable length by extracting n consecutive lines beginning at a random point. If the end of the file is reached before n, the starting point is decremented towards the beginning of the file until all lines can be extracted. If n is greater than the total size of the file, the entire file is used. Results are shown in Figure 9. We see that the optimal attribution accuracy is achieved at a sample length upper bound of 100 LoC. Until this point, the relationship between the LoC upper bound and the attribution accuracy is linear. In a

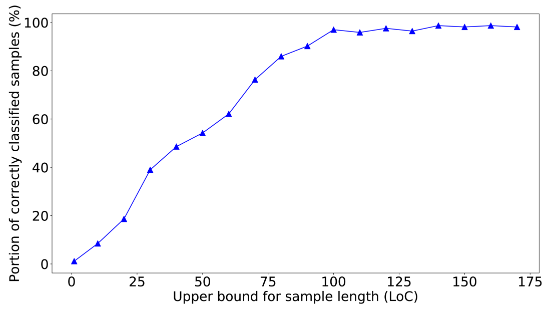

Fig. 9. Performance in the detection of code attribution in GCJ according to the number of LoC per sample.

curated dataset such as GCJ then, the inclusion of samples of length above 100 LoC does not result in improved accuracy.

Second, we evaluate the size in terms of LoC when we work at the *function-level* (i.e., samples are functions as opposed to the entire files or snippet). While in GCJ almost all files are structured in functions, this is not the standard practice in UF. Due to the sparsity of the UF dataset and especially the lengths of the samples after splitting the code into functions, sample lengths are generally not long enough to truncate them like in the first experiment. Thus, we run an experiment using the UF dataset. We take the top 100 authors (those with most number of samples) and get the minimum threshold on the LoC per sample that results in at least 5 remaining samples per author. This threshold is 25 LoC, meaning each sample is at least 25 LoC long. Accordingly, we repeat the experiment increasing this threshold. This way, as the minimum LoC increments, we include progressively fewer of the authors to show the attribution accuracy with a lesser number of authors. Results are shown in Figure 10. We see that a learning task in the UF dataset also achieves its best performance between 90 and 100 LoC.

While the differing methods for the two experiments mean that they can not be compared exactly in a like-for-like manner, both achieve optimal accuracy at the threshold of 100 LoC being used: the first, when samples are at most 100 LoC, the second, when samples

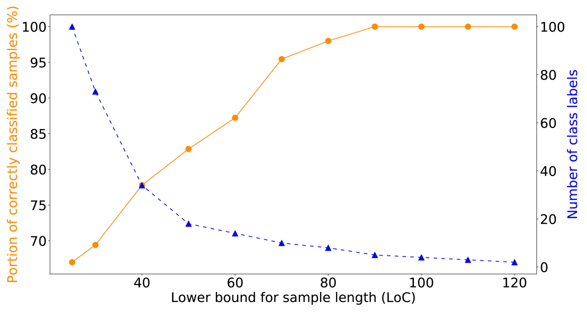

Fig. 10. Effect of discarding low length samples on attribution accuracy on UF function samples.

are at least 100 LoC. The LoC per sample as a heuristic for the amount of stylistic information in a given source code sample provides an indication of when optimal accuracy is achieved. This shows that the number of available data (i.e., LoC per author and per sample) is a relevant difference that affects the accuracy of the classification. We note that the curve in Figure 10 achieves 100% accuracy in a simplified setting where there are only 5 class labels (authors), since only 29 samples have 100 or more LoC. That said, results show that after applying our method, we are able to classify samples of over 100 LoC from these 5 user accounts with 100% accuracy.

Number of samples per author. The intuition is that different samples belonging to the same author will carry stylistic similarities along with differences in functionality and the problem being addressed. As such one might say that the more samples per author, the better a classifier would learn authors' style. Figure 2 shows that this is not necessarily the case. Using the GCJ dataset, where other conditions are kept optimal, the number of samples per author does not have a significant effect on attribution accuracy past 5 samples per author. This is likely due to the wealth of information gathered from each sample, their median length being 68 LoC. The optimal operational setting would have more than 12 samples in UF as evidenced by the continual slight upward trend in Figure 2, however the UF dataset is scarce enough (median of 2 samples per author) that for our purposes we need to draw a threshold at 5 samples and not any higher so that we have enough data for 100 authors.

Number of mislabeled samples. Finally, we modify the dataset by randomly mislabeling a portion of the samples. Under these conditions, changes in the ground truth noise has an expected negative linear relationship with the accuracy of authorship attribution of samples from 98% to 2% for 80% of the ground truth poisoned as shown in Figure 3. This is a powerful effect on overall accuracy, and in our domain we are unable to guarantee ground truth nor measure what portion of our dataset is wrongly labelled in terms of true code authorship.

Take-aways. As discussed in §A, samples in the GCJ dataset have a mean of 74.3 LoC — the largest being 1,184 lines. Meanwhile, in the UF dataset the mean of LoC per sample is 20.9 — the largest being 3,868 lines. The mean number of samples per author is also different (4.5 in UF versus 6.6 in GCJ). As such, after discarding samples in UF below 100 LoC, we see 1,520 authors. However, further removing all authors with less than 5 samples (for which 5-fold cross validation is not possible) we have 52 authors. In contrast, discarding samples below 100 LoC in GCJ we are left with 1,665 authors, and removing all authors with fewer than 5 samples we then retrain 284 authors, over five times the number in UF. This shows that the ideal conditions observed in GCJ in terms of data available are not met in the UF dataset. All in all, we see that it is significantly more challenging to to perform authorship attribution in UF when the number of authors is high. In contrast, when the number of authors is bounded, it is possible to improve accuracy the classifier, with high accuracy when only considering a small number of authors (i.e., less than 20 as shown in Figure 6). This setting may be useful in some investigations where the alleged developer (of a code fragment under scrutiny) has been previously reduced to a small group (e.g., filtering based on geographical location or by the type of activities the users are involved in), by using this in combination with other stylometry approaches, or by focusing on a single forum where few users share code.

C Supplementary Figures

Figure 11 and Figure 12 refer to UF and show the total number of an author's posts plotted against the percentage of these posts that were found to be code clones. Evidently there is no clear pattern, though we note that authors with outlier high numbers of posts tend to have a low percentage of clones. This may be because these users are heavily invested in the community and are more influential actors, often contributing their own code.

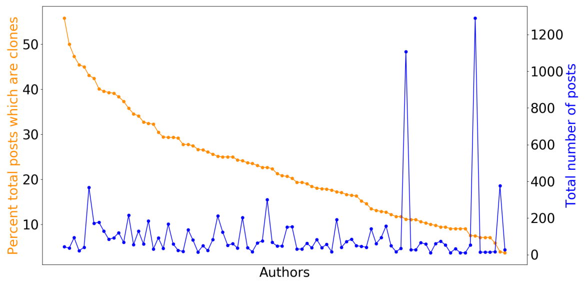

Fig. 11. Posts by users that are clones (%), top 100 users in UF.

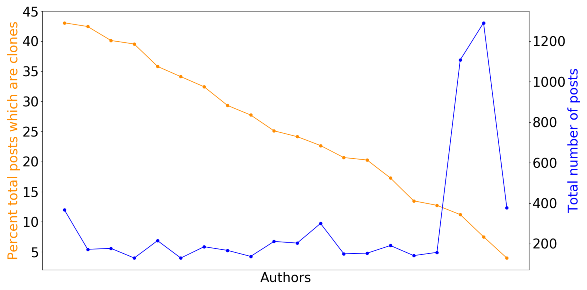

Fig. 12. Posts by users which are clones (%), top 20 users in UF.

D Ethical Issues

In this work, we have used data publicly available for cybercrime-related research by means of an agreement with the Cambridge Cybercrime Centre. We have followed due precautions during the storage and management of the data. Regarding the analyses carried out, our main objective is to analyze the accuracy of existing tools for code stylometry analysis and to provide improvements. Thus we neither aim at the actual identification of the person behind each of these accounts, nor to judge which are the activities carried out on these forums. As mentioned before, our methodology can be viewed as a tool to audit the privacy of online forum users when posting source code, or to help in the identification of these if needed (e.g. during online prosecutions). In this regard, as academics we should remain agnostic of the use made of authorship attribution tools.