# Dataset characteristics for reliable code authorship attribution

Farzaneh Abazari, Enrico Branca, Norah Ridley, Natalia Stakhanova, Mila Dalla Preda

**Abstract**—Code authorship attribution aims to identify the author of source code according to the author's unique coding style characteristics. The lack of benchmark data in the field, forced researchers to employ various resources that often did not reflect real programming practices. Throughout the years, studies used textbook examples, students' programming assignments, faculty code samples, code from programming competitions and files retrieved from open-source repositories. Such data diversity raised concerns about the appropriate data characteristics for reliable evaluation of code attribution.

In this paper, we investigate these concerns and analyze the effect of the dataset characteristics and feature elimination techniques on the accuracy of code attribution. Unlike the majority of the work done in this field which concentrates on designing new features, we explore the nature of data used in the previous studies and assess the factors that influence the attribution task. Within this analysis, we investigate the robustness of three feature sets seen as benchmarks in the attribution research. Based on our findings, we define a process for derive a reduced set of features for accurate and predictable attribution, and make recommendations on the dataset characteristics.

**Index Terms**—Source code attribution, Machine learning, Feature Selection, Authorship attribution, Github

## 1 Introduction

Code authorship attribution techniques aim to identify the author of a given code based on its unique characteristics that reflect an author's programming style. Inspired by social studies in the attribution of literary works, in the past two decades researchers examined the effectiveness of code attribution in the computer security domain. Code attribution techniques have found a wide application in code plagiarism detection [1], [2], [3], biometric research [4], software forensics [5], [6], [7], and analysis of underground actors [8] and malware authorship [9], [10], [11], [12].

Indeed, the research showed that analysis of software might effectively unveil the digital identity of a developer that is manifested through variables, data structures, programming language, employed development platforms, and their settings [9], [13].

Through many research studies that have documented the experiments aimed at studying the effectiveness of code attribution, one question remained common: what is the most appropriate dataset required for attribution of a code? With a lack of benchmark datasets in this field, researchers resorted to using ad hoc data that retain at least some characteristics of programming practices, e.g., archives of students' programming assignments [14], [15], [16], programs written by textbook authors [17], programming competition archives [18], [19], [20], and open-source repositories [21], [22], [23], [24].

Such data diversity in experimentation creates some concerns. Using privately collected data (e.g., students' assignments) hinders reproducibility of the experiments. Open data sources (public repositories and textbook examples) raise questions about the sole authorship and context of considered code. Code received from programming competitions does not reflect personal coding habits due to the artificial and constrained nature of competition setup.

Beyond the concerns about the source of data, datasets vary drastically in size, the number of analyzed authors, samples, length of the code, and even quality of code, making a comparative analysis of studies challenging. For example, MacDonell et al. [6] experimented with 7 authors in the dataset obtaining 88% accuracy in attribution. McKnight et al. [25] achieved 66% with 525 authors. Both Tennyson et al. [17] and Caliskan et al. [18] obtained 98% accuracy while experimenting with 200 samples and 9 samples per author, respectively. Questioning the length of available code, Dauber et al. [26] showed that it is possible to attribute code snippets that have only a few lines of code, while the majority of studies found it necessary to experiment with lengthy code.

Such ad-hoc approaches to data selection naturally prompt questions regarding the appropriate data characteristics for code attribution:

- How much data is enough to demonstrate the effectiveness of the attribution method? In other words, what are the suitable data characteristics such as the number of authors, samples per author, and length of samples necessary for the code attribution task?
- Are these characteristics dependent on the data source, data quality, and context? In other words, would these data characteristics be appropriate for data retrieved from different environments.
- What is the effect on the accuracy of code attribution if appropriate characteristics are not selected?

The concerns regarding appropriate data characteristics are not new. The Burrows et al. [27] study was one of the first to question the quality and variability of data. A few other researchers also brought up potential effects of data bias on study results [26], [28], [29], [30].

Answering the questions regarding the appropriate data characteristics is essential for software attribution domain and may have implications for both accuracy and performance of attribution techniques in the deployment environment. Within this context, it is necessary to understand whether the results obtained by previous studies were due to the optimal, or perhaps coincidental, match between data and approach, or due to the attribution technique's tolerance to dataset imperfections. This is the focus of our work.

We approach these questions by first analysing data employed by the existing studies in the area of source code attribution and replicating their experiments. We proceed with exploring the effect of common code, analysing source code size, samples size, and author set size effects on the attribution accuracy of different classifiers. Finally, we develop a balanced large-scale authorship attribution dataset and offer it to the research community.

This article is organized as follows. In Section 2, we provide a summary of previous studies. Section 3 describes datasets characteristics and feature sets that are used by recent works. We explore the effects of preprocessing and feature extraction on the accuracy of different classifier in Section 4. Section 5 explores the effect of source code size, sample size, and author set size, respectively on attribution accuracy. Finally, we summarize our findings in Section 6.

## 2 Related work

The cornerstone of any code authorship attribution study is the dataset. Over time the researchers experimented with data that can be broadly categorized in several groups: academic data, data obtained from open source repositories (OSS), and mixed data retrieved from both open source projects and academic studies. The summary of the reviewed studies is given in Table 1.

### 2.1 Academic data

In the late 1990s, researchers mostly used archives of students' programming assignments [14], [15], [16]. Sharing this data was challenging due to privacy concerns. If such data was allowed to be shared, it was typically heavily anonymized to remove identifiable information commonly used for authorship attribution.

The first in-depth study of source code authorship attribution was performed by **Krsul et al. [14]**. The researchers leveraged C programs collected from students, faculty, and staff to study the impact of programming structure, programming style and layout on attribution. This study experimented with over 20 different classification methods on a small set of 88 programs with a variable success.

Student-submitted programs were also used in the study by **Elenbogen et al. [16]**. A total of six simple metrics based on heuristic knowledge and personal experience were considered: number of lines of code, number of variables, number of comments, variable name length, looping constructs, and number of bits in the compressed program. The attribution analysis was able to achieve results similar to those obtained by Krsul et al. [14].

**Burrows et al. [35]** explored the accuracy of various similarity measures (e.g., cosine, Dirichlet) for attribution of 1597 student assignments collected for a period of over 8 years. Within this study, the authors examined six groups of features that, in their opinion, should represent good programming style: white space, operators, literals, keywords, I/O words, and function words.

### 2.2 Open-source repositories and mixed data

With the growing popularity of open-source repositories, researchers have started leveraging this source of data. **Frantzeskou et al. [15]** employed source code samples from the FreshMeat repository[1]. The researchers introduced the SCAP (Source Code Author Profiles) method that represents an author's style through byte-level $n$-gram profiles. Since an author profile is comprised of a list of the $L$ most frequent $n$-grams, several follow-up studies were performed to determine the best values of $L$ and $n$ ([40], [41], [15], [42], [43], [44], [45]). The best results were achieved for values of $L$ equal to 1500 or 2000 and $n$-gram sizes of 6 or 7. Table 1 lists three main experiments performed in this study [15].

Several studies worked with free software projects hosted on SourceForge[2] [34], [24], [23]. **Lange et al. [34]** presented a method involving the similarity of histogram distributions of code metrics, which were selected using a genetic algorithm. The authors formulated 18 layout and lexical metrics as histogram distributions. **Shevertalov et al. [24]** focused on improving classification accuracy through discretization of four metrics (leading spaces, leading tabs, line length, and words per line). Similar to Lange et al. [34] and Shevertalov et al. [24] studies, **Bandara et al. [23]** used a combination of layout and lexical features for attribution. **Gull et al. [21]** employed a new characteristic of the coding style of the programmer: code smell. Code smell is a known phenomena in software engineering domain that usually manifests a bad programming practice or design problem. As such the presence of certain code smells can be indicative of a certain programming style.

Similar to other studies, **Zhang et al. [22]** leveraged layout characteristics for attribution. The programmer's profiles were constructed based on four feature categories: layout (e.g., whitespaces), style (e.g., number of comments), structure (average line length), and logic (character level n-grams). The authors experimented with several algorithms, yet the best result was obtained with SVN (83.47%).

Most of the authorship attribution techniques extract features from the source code such as variable names and keyword frequencies. Instead of using layout and lexical features, **Pellin et al. [31]** used only syntactic features derived from the abstract syntax (AST) trees. The approach was unique; instead of creating frequency-based feature vectors (common in attribution), the authors fed the AST directly to the tree-based kernel machine classifier. Unfortunately, the analysis with only 2 authors does not provide sufficient ground for comparison with other methods.

**Kothari et al. [33]** explored attribution based on programmers' profiles built with two types of metrics: style characteristics (such as distributions of leading spaces, line size, etc.), and character n-grams. Although the second set was similar to the one introduced by Frantzeskou et

1. http://freshmeat.sourceforge.net/
2. https://sourceforge.net/

TABLE 1
Previous studies in source code authorship attribution.

| Related Work | Year | # Authors | Samples/ author | Tot. Files | LOC | Avg. LOC | # Features | Features Type | Lang. | Dataset | Method | Result |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Krsul et al. [14] | 1997 | 29 | - | 88 | - | - | 49 | lay,lex | C | academic | Discr. analysis | 73% |
| MacDonell et al. [6] | 1999 | 7 | 5-114 | 351 | 16-1480* | 182* | 26 | lay,lex | C++ | mixed | CBR, NN | 88% |
| Pellin et al. [31] | 2000 | 2 | 1360-7900 | - | - | - | - | synt | Java | multiple OSSs | SVM | 73% |
| Ding et al.[32] | 2004 | 46 | 4-10 | 259 | 200-2000 | - | 56 | lay,lex | Java | mixed | Discr. analysis | 67.2% |
| Frantzeskou et al. [15] | 2006 | 30 | 4-29* | 333* | 20-980* | 172* | 1500 | lex | Java | FreshMeat (OSS) | SCAP | 96.9% |
| Frantzeskou et al. [15] | 2006 | 8 | 6-8 | 60* | 36-258 | 129 | 2000 | lex | Java | academic | SCAP | 88.5% |
| Frantzeskou et al. [15] | 2006 | 8 | 4-30 | 107 | 23-760 | 145 | 1500 | lex | Java | FreshMeat (OSS) | SCAP | 100% |
| Kothari et al. [33] | 2007 | 8 | - | 220 | - | - | 50 | lex | - | academic | NaiveBayes | 69% |
| Kothari et al. [33] | 2007 | 12 | - | 2110 | - | - | 50 | lex | - | multiple OSSs | NaiveBayes | 61% |
| Lange et al. [34] | 2007 | 20 | 3 | 60 | 336-80131* | 11166 | 18 | lay,lex | Java | SourceForge | Genetic alg. | 55.0% |
| Elenbogen et al. [16] | 2008 | 12 | 6-7 | 83 | 50-400* | 100* | 6 | lay,lex | C++* | academic | Decision tree | 74.70% |
| Burrows et al. [35] | 2009 | 10 | 14-26 | 1597 | 1-10789 | 830 | 325 | lex | C | academic | ranking | 76.78% |
| Shevertalov et al. [24] | 2009 | 20 | 5-300 | - | - | 11166 | 163 | lay,lex | Java | SourceForge | Genetic alg. | 54.3% |
| Bandara et al. [23] | 2013 | 10 | 28-128 | 780 | 28-15052 | 44620 | 9 | lay,lex | Java | SourceForge | Regression | 93.64% |
| Bandara et al. [23] | 2013 | 9 | 35-118 | 520 | 20-1135 | 6268 | 9 | lay,lex | Java | academic | Regression | 89.62% |
| Tennyson et al. [17] | 2014 | 15 | 4-29 | 7231 | 1-3265 | 50 | - | lex | C++, Java | mixed | Bayes Classifier | 98.2% |
| Caliskan et al. [30] | 2015 | 250 | 9 | 2250 | 68-83 | 70 | 120000 | lay,lex,synt | C++ | GCJ | RF | 98.04% |
| Caliskan et al. [18] | 2015 | 1600 | 9 | 14400 | 68-83 | 70 | - | lay,lex,synt | C++ | GCJ | RF | 92.83% |
| Wisse et al. [36] | 2015 | 34 | - | - | - | - | 1689 | lay,lex,synt | JS | GitHub | SVM | 85% |
| Yang et al. [37] | 2017 | 40 | 11-712 | 3022 | 16-11418 | 98.63 | 19 | lay,lex,synt | Java | GitHub | PSOBP | 91.1% |
| Alsulami et al. [19] | 2017 | 10 | 20 | 200 | - | - | 53 | synt | C++ | GitHub | NN | 85% |
| Alsulami et al. [19] | 2017 | 70 | 10 | 700 | - | - | 130 | synt | Python | GCJ | NN | 88.86% |
| Gull et al. [21] | 2017 | 9 | - | 153 | 100-12000 | - | 24 | lay, lex | Java | PlanetSource Code | NaiveBayes | 75% |
| Zhang et al. [22] | 2017 | 53 | - | 502 | - | - | 6043 | lay,lex | Java | PlanetSource Code | SVM | 83.47% |
| Dauber et al. [26] | 2017 | 106 | 150 | 15900 | 1-554 | 4.9 | 451368 | lay,lex, synt | C++ | GitHub | RF | 70% |
| McKnight et al. [25] | 2018 | 525 | 2-11 | 1261 | 27-3791* | 336* | 265 | lay,lex, synt | C++ | GitHub | RF | 66.76% |
| Simko et al. [20] | 2018 | 50 | 7-61* | 805 | 10-297* | 74* | - | lay,lex, synt | C | GCJ | RF | 84.5%* |
| Abuhamad et al. [38] | 2018 | 8903 | 7 | 62321 | - | 71.53 | - | lex | LO | GCJ | RNN, RF | 92.30% |
| Ullah et al. [39] | 2019 | 1000 | - | - | - | - | - | lex, CFG | LO | GCJ | NN | 99% |

- No data
* Data obtained from personal communication
^ LO: language-oblivious approach

al. [41], Kothari et al. used entropy to identify the fifty most significant metrics for each author. In all experiments, the 4-gram frequencies outperformed the other six metrics. Later, the study by Burrows et al. [27] indicated that this method [33] outperformed the SCAP method proposed by Frantzeskou [42].

Studies by MacDonell et al. [6], Ding et al. [32], and Tennyson et al. [17] used the combination of open source projects and academic code.

Overall, open source repositories have a clear advantage of offering a rich and diverse pool of programmers and source code for experiments. Yet a lack of clear distinction among programs written by multiple authors presents a major challenge in attribution analysis.

### 2.3 GoogleCodeJam data

The majority of recent attribution studies [18], [19], [20] have used programs developed during the GoogleCodeJam,[3] an annual international coding competition hosted by Google. Given a set of problems, the contestants are asked to provide solutions in a restricted time. The availability of statistical information, such as the popularity of programming language, contestants' skill levels, and their nationalities make data from the GoogleCodeJam especially useful for authorship profiling. For authorship attribution, use of this data has been extensively criticized mostly owing to its artificial setup [26], [29], [30]. The researchers argued the existing competition setup gives little flexibility to participants resulting in somewhat artificial and constrained program code.

**Caliskan et al. [18]** was the first study to showed that attribution can be successful on large-scale datasets. The dataset collected from the GoogleCodeJam competition included 14,400 files from 1600 authors. Another difference between Caliskan et al. study and the earlier works is the composition of their feature set that included lexical, layout and syntactic features. While the lexical and syntactic categories accounted for the majority of features, the total feature set represented 120,000 dimensions. **Simko et al. [20]** replicated the Caliskan et al. [18] approach on smaller dataset. As opposed to the original study [18] that achieved nearly 99%, Simko et al. was only able to accurately attribute 84.5% samples of 50 authors.

To avoid hand-tuned feature engineering, **Alsulami et al. [19]** employed a deep neural networks model that automatically learned the efficient feature representations of AST. The authors used Long Short-Term Memory networks (LSTM) for tree extraction and experimented with several different classification algorithms. In many respects, Alsulami et al. study is similar to work done by Pellin et al. [31], who employed AST features for attributing authors using a tree-based kernel SVM algorithm.

GoogleCodeJam data was also used in adversarial attribution of source code by Quiring et al. [46] and Matyukhina et al. [47].

3. https://code.google.com/codejam/

### 2.4 GitHub (OSS) data

Given the criticism of constructed datasets , several researchers ventured to collect data "in the wild" [26], [37], [36], [25]. The majority of them used repositories found on GitHub[4]. In addition to the presence of significant noise in the data (e.g., junk code), the main concern that remained during their analysis was the sole authorship of the extracted code. Similar to other open-source repositories, GitHub does not offer reliable facilities to differentiate code written by multiple authors.

All previous studies focused on attribution of complete programs. **Dauber et al. [26]** were the first to analyze short, incomplete, and sometimes uncompilable fragments (some samples containing just 1 line of actual code).

GitHub was also the source of data in **Yang et al. [37]** study. The majority of lexical and layout metrics employed in this study were derived from Ding et al. [32] feature set.

The **Wisse et al. [36]** work focused on identification of the JavaScript programmers with the aim to identify writers of web exploit code. Similar to Caliskan et al. [18] study, the authors used features derived from AST with a combination of layout and lexical features to describe the coding style of the author. However, Wisse et al. enriched the AST features with character and node n-grams instead of adding word unigram features as it was done in Caliskan et al. [18] study. Although both methods were language-dependent, many proposed features can be mapped to other languages.

Looking at adversarial attribution, **McKnight et al. [25]** developed a technique to assist programmers in hiding their coding style and consequently preventing their code attribution. Similar to Wisse et al. [36] study, the researchers used node frequencies derived from AST. However, they further enriched this information with node attributes, identifiers, and comments. A different approach was taken by **Abuhamad et al. [38]**. To avoid uncertainties of feature engineering the authors used deep learning architecture with a recurrent neural network (RNN). The approach was explored on several languages (C, C++, Java, Python) and only leveraged lexical features. With exception of the Caliskan et al. [18] study, this was the only work that ventured to attribute authors on a large scale (8,903 authors).

Table 1 gives a summary of all source code attribution studies that conducted experiments. A quick analysis of attribution studies shows that in most cases researchers who used the GoogleCodeJam dataset [18], [20], [19] in their experiments received much better accuracy in attribution task than those who chose other data sources. For example, SourceForge data on average showed much less accuracy than academic data. While experiments with GitHub data in almost all cases gave much lower accuracy than with GoogleCodeJam programs. Such difference in accuracy was also noted by Dauber et al. [26] and Alsulami et al. [19]. The authors showed that using the Caliskan et al. [18] method on the GitHub dataset considerably decreases accuracy, from 96.83% (GoogleCodeJam) to 73% (GitHub) and from 99% to 75.90%, respectively.

Given the considerable variability of data characteristics of the reviewed studies, it is essential to understand the factors that influence the attribution results.

4. https://github.com/

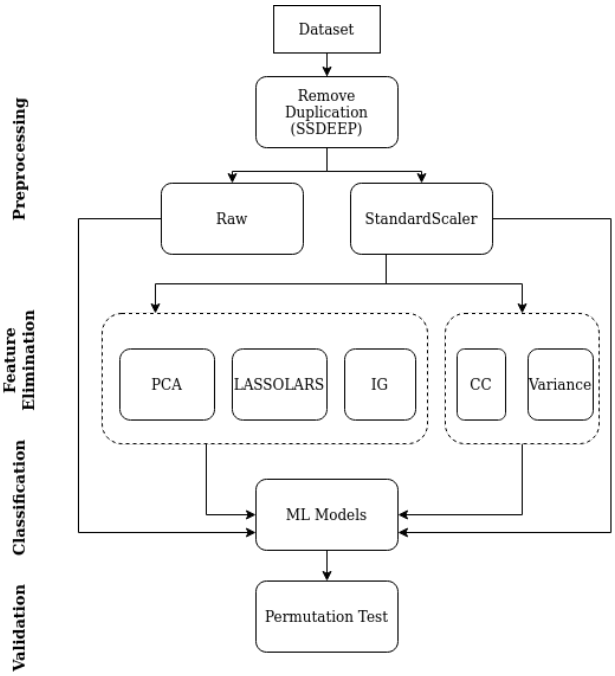



Fig. 1. Flow of the analysis with multi-level feature elimination approach

## 3 METHODOLOGY

Figure 1 illustrates the flow of our analysis that consists of four stages: *preprocessing* to reduce duplicated code and rescale features, *feature elimination* to reduce noise and select optimal set of features for analysis, *classification*, and *validation* to evaluate the generalization power of the classification model and assess its dependency on the features.

### 3.1 Datasets

In our analysis we used data obtained from the previous studies, from the GitHub repository and from the GoogleCodeJam programming competition. The details of these datasets are given in Table 2.

**Datasets from previous studies**. We requested original datasets employed by the previous studies and replicated the analysis on the three sets that we received. These datasets are Simko et al.[20], Dauber et al. [26] and McKnight et al.'s [25] studies.

These datasets represent diverse experimental data quality used in attribution, they are derived from two different sources: GitHub repository, and textbook programming assignments, and include code written in C and C++ programming languages. For discussion purposes, in the rest of this paper, we will refer to these datasets by the name of the first author, e.g., the 'McKnight dataset' will refer to data from the work by McKnight et al. [25].

**GitHub Repository**. GitHub, an open-source software development platform, contains nearly 67 million repositories. The programs in GitHub are typically more complex, include third-party libraries, several encodings, use source code from other authors and focus on solving diverse tasks (e.g., from game development to middleware).

We collected programs from 2018 to 2020 by using the Github API. Although it is difficult to guarantee sole

authorship of any code posted online, we took reasonable precautions by filtering repositories marked as forks, as these are typically copies of other authors repositories and do not constitute original work. An additional check for multiple-author repositories was performed by examining the commit logs. Repositories with logs containing more than one unique name and email address combination (potentially indicating an involvement of several authors) were also excluded. For this analysis, we collected source code of over 380,000 programs written in Java programming language from 3905 authors.

We offer our dataset to the research community in the hope of diversifying and strengthening experiments in this field [5].

**GoogleCodeJam Dataset**. Given the extensive criticism of using data extracted from GoogleCodeJam programming competition, for our analysis we also assembled a dataset containing code from the 2008 to 2018 competitions. The overall dataset contains around 13,000 authors with java samples. Similar to the previous studies, we use a subset of GCJ dataset. The extracted dataset contains 25,825 programs from 1033 authors written in Java programming language. We refer to it as *GCJ* dataset.

### 3.2 Preprocessing

One of the characteristics of datasets that might influence the results of the analysis is the presence of noise and duplicated code. We filter our datasets to remove unparseable samples, samples with less than 4 characters and authors with less than 5 samples.

The cleaning of data also includes removal of third-party data, including shared libraries and duplicated code. The existence of such third-party code contributes to the amount of noise in a data which might effectively make the attribution process more challenging [25]. Indeed, third-party libraries are typically shared among software programs, and therefore, exhibit a different coding style that might confuse the attribution process.

On the other hand, duplicate code samples belonging to the same authors introduce bias in classification (i.e., giving more weight to repetitive style features) and make the attribution performance in different deployment environment unpredictable. The situation becomes more interesting when code samples are not identical, but slightly different (we refer to it as common code). This often happens with code reuse, i.e., software developers commonly reuse code previously written by them or in some instances by others. To understand the influence of duplicate and common code on authorship attribution accuracy, we explore its presence in our input datasets.

We employ context-based fuzzy hashing to measure the similarity between programs within one author and between authors in datasets. We use Context Triggered Piecewise Hash (CTPH) approach introduced by Kornblum [48] that allows to detect identical and partial similarity in code. The implementation of this approach is known as ssdeep.

5. https://cyberlab.usask.ca/authorattribution.html

As it is illustrated in Figure 2, the percentage of similarity among datasets varies significantly. Dauber and Simko datasets have the most similarity among the datasets. At 75% threshold only around 40% of code remains in the Simko dataset. However, Github and GCJ sets show less similarity as the number of files retained in the dataset after removing similarities are 89.8% and 99.6%, respectively. To balance the amount of data available for analysis, we strategize, i.e., between two authors with a similar file we remove a file from an author with more available code samples. This allows us to retain more number of authors irrespective of the similarity threshold.

In general, a lower threshold leads to less tolerance towards common code, so more files are removed. The overall trend in our datasets is stable (except the Simko's data) and the amount of removed data for any threshold above 75% is similar. We thus error on the conservative side, and retained files with less than 75% of code duplication. The statistics of the dataset after removing code duplication is given in Table 2.

### 3.3 Feature Standardization and Elimination

The majority of the reviewed studies claim to offer unique sets of features capable of capturing developers' stylistic traits. Since the accuracy of attribution changes drastically depending on the study, we aim to explore the impact of features set on attribution and the extent of their contribution to the final accuracy of the approach.

Our review shows that the existing studies experiment in source code attribution authorship belong to one of the following groups: layout, lexical, or syntactical features.

- **Layout features** refer to format or layout metrics that describe the appearance of the code (e.g., average line length, the number of spaces) and the number of specific characters (e.g., commas, curly brackets).
- **Lexical features** are divided into the metrics of programming style (e.g., author's preference to use the short or long name of classes and methods, the type of branching statements), and the frequency of n-grams. Style features are programming language dependent, while n-grams features, based on sequences of n-characters extracted from the source code, are not.
- **Syntactic features** represent the code structure that is resistant to the changes in the code layout. A abstract syntax tree (AST) is a common way to determine how a code is structured.

For our analysis, we therefore select the feature sets that represent these groups.

The first studies in the code authorship attribution primarily experimented with numeric features extracted from layout and lexical levels. The study by Krsul et al. [14] was first to publish 49 metrics for C/C++ language. MacDonell et al. [6] and Gray et al. [49] further improved them by using ANOVA test to measure features' significance. Ding et al. [32] combined and adapted these metrics for the Java language. Although the authors never provided the final subset in the original study, this was corrected by the follow-up study by Burrows et al. [27] which derived a final feature

TABLE 2
The employed datasets' statistics

| Dataset | Language | Num. Authors | Num of Files | Range Samples/ Author | Range LOC | Avg. LOC | Range char. per line | Avg char. per line |
|---|---|---|---|---|---|---|---|---|
| Github | Java | 3,905 | 380,732 | 4-11,889 | 1- 67,229 | 170.67 | 1-173,674 | 33.41 |
| GoogleGodeJam (GCJ) | Java | 1,033 | 25,825 | 25 | 30-300 | 99.13 | 1-36,871 | 25.36 |
| McKnight [25] | C | 676 | 40,246 | 6-1,437 | 1-110,860 | 336.34 | 1-76,041 | 28.28 |
| Dauber (Snippets) [26] | C++ | 62 | 201,060 | 6 - 48,725 | 3 -17,759 | 13.98 | 1 - 15,326 | 28.77 |
| Simko [20] | C | 58 | 2,838 | 14-366 | 10-1,256 | 89.71 | 1-294 | 19.31 |
| *The datasets' statistics after preprocessing* | | | | | | | | |
| Github | Java | 3,128 | 173,919 | 5-3,669 | 1-67,229 | 139.84 | 1-173,674 | 32.17 |
| GoogleGodeJam (GCJ) | Java | 1,033 | 25,723 | 15 - 25 | 30-300 | 99.04 | 1-36,871 | 25.36 |
| McKnight [25] | C | 675 | 36,854 | 6-804 | 1-110,860 | 345.81 | 1-76041 | 28.33 |
| Dauber (Snippets) [26] | C++ | 38 | 135,883 | 5-36,680 | 4-998 | 14.5 | 1-15,326 | 26.99 |
| Simko [20] | C | 46 | 1134 | 14-70 | 13-502 | 93.15 | 1-241 | 19.34 |

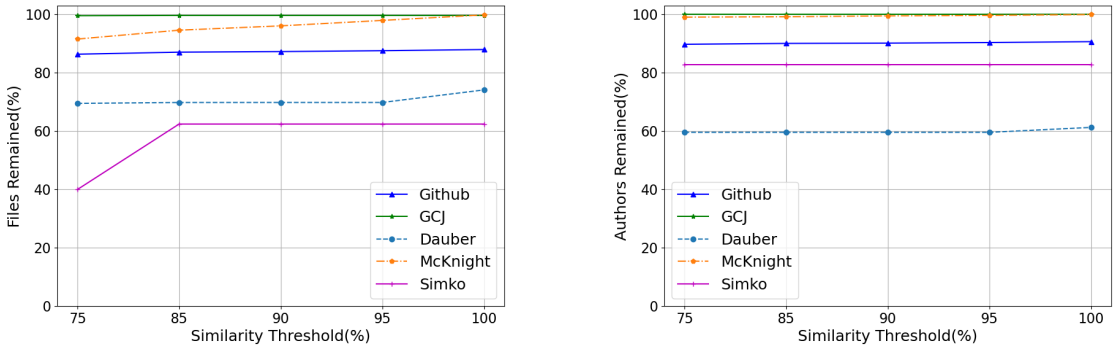



Fig. 2. Effects of removing similar code on number of files and authors

set of 56 metrics from the original Ding et al. [32] study. Lange et al. [34], Gull et al. [21], Elenbogen et al. [16], and Yang et al. [37] also experimented with Ding et al. [32] features in their works.

Throughout the evolution of attribution domain, n-gram features attracted a lot of research interest due to their simplicity and generally good performance [35]. The Kothari et al. [33] study featured a set of n-grams that was later leveraged by other researchers (e.g., Bandara et al. [23], Shevertalov et al. [24], Franzeskou et al. [15], Tennyson et al. [17], and Zhang et al. [22]).

The final feature set that we selected for experiments is a set derived by the Caliskan et al. study [18]. This set contains syntactic features derived from AST along with some layout and lexical features. Dauber et al. [26], Simko et al. [20], and McKnight et al. [25] used these features in their studies.

We use three prominent feature sets for evaluation: Ding et al. [32], Kothari et al. [33], and Caliskan et al. [18].

These selected feature sets are the most prominent in the history of source code attribution, widely used and are often seen as benchmark feature sets. In the rest of this work, we will refer to these sets as the Ding's, Kothari's, Caliskan's features, respectively.

### 3.4 Feature selection methods

The goal of feature selection task is to select the subset of features useful or relevant for building a good classifier. Depending on the goal, the features might be selected based on their usefulness to a specific classifier or their overall ranking irrespective of the algorithm [50]. Since in this work we experiment with various classification algorithms, we employ the following feature selection methods:

**Standardization**. The preliminary analysis of datasets show that the derived features vary drastically in scale, have a high percent of missing values, and have features with low variability. Removing such variation using feature scaling through standardization is one of the necessary steps that can remove the bias (i.e, domination of some features) and allow the classification algorithm to learn properly. Yet this step is never mentioned in attribution, thus we also train our classifiers on raw data for comparison. We standardize features by removing the mean and scaling them to unit variance.

**Variance analysis**. Features with low variance ($var = 0$) should be removed as they are unlikely to contribute to classification model.

**Cross correlation analysis**. In addition to variability in accuracy and scale, the three feature sets employed in our analysis differ significantly in the number of features. Ding et al. [32] used discriminant analysis to identify 56 of the best performing features as a result, the number of features are constant and does not depend on the size of data. Kothari et al. [33] selected 50 top features for each author. While, Caliskan et al. [18] offered a diverse set of features that depended on the analyzed data. None of the studies offered a clear analysis of selected features and rationale for their selection aside from overall method performance.

The preliminary analysis of these features sets revealed

co-dependency of some features, e.g., term frequency (TF) of AST node types and term frequency-inverse document frequency (TFIDF) of AST node types in the Caliskan's feature set. Correlated features are likely to carry redundant and useless information. We thus venture to conduct correlation analysis to determine the extent to which two or more features depend on each other.

We calculate Pearson's correlation coefficient $\rho$ to represent the statistical relationships between features. The correlation coefficient ranges from $-1$ to 1. A positive correlation indicates that the variables increase or decrease together, while a negative correlation indicates the inverse relationship.

**LassoLars**. Least Absolute Shrinkage and Selection Operator (LASSO) technique is commonly used for high-dimensional data. LassoLars is a Lasso model implemented using the Least-angle Regression (LARS) algorithm. The algorithm attempts to determine features that are the most correlated with the target. When multiple features have equal correlation, LassoLars proceeds in a direction equiangular between the features [51].

**Principle component analysis** (PCA), is a widely used dimensionality reduction technique that detects features that are statistically significant. Using PCA, we can find a list of features that express a linear relationship [52].

**Information Gain**. Among the reviewed studies, information gain was the most prevalent method for feature selection. One of its criticism is that it leads to the selection of redundant features and similar performance might be obtained with smaller subset of features [50].

### 3.5 Classification algorithms

Similar to variability in data characteristics, previous studies in source code authorship attribution employed various classification algorithms for attribution analysis including Random Forest [18], [26], [25], [20], Naive Bayes [33], [21], Decision tree [16], Neural network [37], [6], [19], Regression analysis [23], Support vector machines [36], [31], [22] and Discriminant analysis [14], [32].

The selection of a classification algorithm is typically driven by the nature of the data and the set of features employed for analysis. Algorithm selection is also influenced by a number of other characteristics such as the algorithm's capability to cope with scarce or voluminous data and the algorithm's sensitivity and ability adapt to the changes. A comprehensive analysis of data should indicate the effectiveness of an algorithm's behaviour in a deployment setting and its ability to produce expected results. Yet most studies fail to justify their choice of a classification algorithm. This lack of justification leads to uncertainty about what algorithms are the most suitable for subtleties of the code attribution domain.

In our work, we explore performance of 7 classification algorithms previously employed by code attribution studies:

**Gaussian Naïve Bayes** (GNB) is based on Bayes theorem that assumes an independence between features [53]. Even though feature independence assumption rarely holds true, NB models perform surprisingly well in practice [54]. Naïve Bayes classifiers are fast compared to more sophisticated methods and require a small amount of training data to estimate the necessary parameters. The Gaussian Naïve Bayes classifier is one of its versions that follows a Gaussian distribution and assumes the presence of data with continuous values which is the case in our datasets.

**Neural Network** (NN) [55] are a series of algorithms that mimic the operations of a human brain to detect relationships between high volumes of data. Since neural networks can have many layers and parameters with non-linearities, they are very effective at modelling highly complex non-linear relationships. Neural networks operate well with large amounts of training data.

**Decision Trees** (DT) [56] produces a sequence of rules that can be used to classify the data when a data of features together with its target are given. The decision tree classifier is a discriminative model. It can be unstable because small variations in the data might result in a completely different tree being generated. There is a high probability for overfitting the model if we keep on building the tree to achieve high purity.

**Discriminant Analysis** [57] builds a predictive model for authors' coding style. The model is composed of discriminant functions based on linear combinations of the features that provide the best discrimination between the authors. The functions are generated from a sample of source code for which author is known; the functions can then be applied to new source code that have measurements for the predictor features but have unknown author. This model assumes that different classes generate data based on different Gaussian distributions. To train a classifier, the fitting function estimates the parameters of a Gaussian distribution for each class and Gaussians for each class are assumed to share the same co-variance matrix. We employ a Linear Discriminant Analysis (LDA) version of DA that provides linear decision boundaries.

**Support vector machine**(SVM) [58] is a representation of the training data as points in space separated into categories by a clear margin that is as wide as possible. Wider margins mean more informative features. Unclassified data are mapped into that same space and predicted to belong to a class based on which side of the gap they fall. There are four main types of kernel: polynomial, RBF, Sigmoid and linear. Non-linear models are especially useful when the data-points are not linearly separable. We use the Linear SVC implementation of SVM that supports the linear kernel.

**Random forest** (RF) [59] classifier is an ensemble that fits a number of decision trees on various sub-samples of datasets and uses the average to improve the predictive accuracy of the model. The RF algorithm builds trees from a sample drawn with replacement from the training set. The selected split is the best split among a random subset of all the features. The result of this randomness increases the bias of the forest, however, due to the averaging, the variance decreases, which compensates for the increased bias and generates a better model.

**Logistic Regression**(LR) [60] is a linear classifier that predicts probabilities rather than classes. We use a multinomial logistic regression classification to calculate the probability of a source code belonging to an author. The logistic regression classifier works well for predicting categorical outcomes. However, it requires that each data point be

TABLE 3
The parameters of the classification algorithms

| Alg. | Parameter | Kernel |
|---|---|---|
| GNB | var_smoothing = 1e-9 | Non-linear |
| NN | max_iter=10000, learning_rate='adaptive', solver='adam', alpha=1 | Non-linear |
| DT | max_depth=100 | Non-linear |
| LDA | solver=svd, shrinkage=None | Linear |
| SVC | kernel="linear", C=0.025 | Non-linear |
| RF | n_estimators=100, min_samples_split=2, min_samples_leaf = 1, max_features="log2", criterion='entropy' | Non-linear |
| LR | penalty="l2", max_iter=100000, solver="lbfgs", multi_class="multinomial" | Linear |

independent, in essence it attempts to predict outcomes based on a set of independent variables.

**Experimental parameters** All analysis was implemented using the Python language (v 3.8.4) with the scikit-learn library (v 0.23.1). A summary of the classification algorithms' parameters is given in Table 3. 5-fold cross-validation is employed to measure the accuracy of the machine learning model. All experiments were performed on an Ubuntu server equipped with 384 GB of RAM and 32 CPU cores.

### 3.6 Permutation Test

As the last step in our analysis, we employ permutation analysis as a model-agnostic method to estimate the quality of the reduced feature sets for attribution tasks "in the wild". This test measures how likely a given classifier would obtain this accuracy with a given set of features by chance. In the recent years, a number of studies have suggested to employ the permutation test to assess the performance of the classifier and the significance of the selected features [35], [61].

This technique is based on repeatedly shuffling the values of each feature in random order and then running a classification model against the new feature set. If the model depends on the shuffled feature, then the resulting prediction can cause a higher error, consequently decreasing an overall accuracy. To measure the importance of the shuffled features, we employ p-value. The p-value approximates the probability that the reduced feature's accuracy would be obtained by shuffled features (Equation 1).

$$p - value = \frac{C + 1}{n_{permutations} + 1} = \frac{1}{101} = 0.0099 \quad (1)$$

where $C$ is the number of permutations which accuracy is greater than the reduced feature set's accuracy.

## 4 Analysis Results

To examine the impact of the datasets' characteristics and related decisions that researchers take during the validation of code attribution techniques, we perform two sets of experiments. *First*, we explore the effects of preprocessing and feature elimination on the accuracy of different classifiers compared to the baseline evaluation of the selected feature sets. *Second*, based on the performance result of different datasets, we measure the importance of the datasets' characteristics.

### 4.1 Baseline evaluation

To create a baseline for our analysis, we apply three selected feature sets on five datasets. Table 4 shows the obtained results on these datasets. Due to the scale, Caliskan's features cannot be used on any of the datasets except Simko's data. This is somewhat expected, as the authors noted that their approach cannot be used without feature elimination.

As the results show, none of the feature sets can adapt to the changes in data, which essentially indicates that in a deployment setting none of the techniques will produce expected results. Although the accuracy of attribution with every feature set varies significantly among datasets, the Kothari features show the highest accuracy across different datasets. Their originally reported results ranged from 61% to 69% accuracy. We obtained even higher results on Simko (99%) and GCJ (97%) datasets. Dings et al. [32] originally obtained attribution accuracy of 67%, we were able to achieve 63% on Simko's data. As we mentioned earlier, using Github dataset considerably decreases accuracy for all attribution approaches. These results provide us a baseline for further analysis (referred to as *BASE*).

TABLE 4
The baseline accuracy results

| Features | Classif. | Dataset | | | | |
|---|---|---|---|---|---|---|
| | | Githib | GCJ | McKnight | Dauber | Simko |
| Ding features | #features | 58 | 58 | 58 | 58 | 58 |
| | GNB | 1% | 24% | 2.8% | 0.4% | 12% |
| | NN | 12% | 46% | 20% | 33.5% | 46% |
| | DT | 23% | 43% | 31% | 36.5% | 33% |
| | LR | 23% | 55% | 24% | 33% | 42% |
| | RF | ME | 56% | 42% | 36% | **63%** |
| | LDA | 13% | 41% | 16% | 28% | 35% |
| | SVC | NC | 45% | NC | 38.5% | NC |
| Caliskan features | #features | 27,932,807 | 720,478 | 4,038,601 | 627,834 | 36,917 |
| | GNB | ME | ME | ME | ME | 82% |
| | NN | ME | ME | ME | ME | 87% |
| | DT | ME | ME | ME | ME | 82% |
| | LR | ME | ME | ME | ME | 83% |
| | RF | ME | ME | ME | ME | 85% |
| | LDA | ME | ME | ME | ME | 2% |
| | SVC | ME | ME | ME | ME | NC |
| Kothari features | #features | 2388 | 760 | 2007 | 1065 | 609 |
| | GNB | 45% | 92% | 55% | 32% | 95% |
| | NN | 27% | 93% | 19% | 46% | 98% |
| | DT | 29% | 95% | 30% | 43% | 98% |
| | LR | ME | 93% | 51% | 47% | 98% |
| | RF | ME | **97%** | 52% | 47% | 98% |
| | LDA | NC | 83% | 36% | 39% | **99%** |
| | SVC | NC | 90% | 31% | 36% | 29% |

NC: models do not converge for the combination of the given features
ME: memory error due to insufficient system memory

### 4.2 Attribution across Feature Elimination

Cross correlation step is performed on the original feature sets after standardization. Figure 3 shows the percentage of correlated features with respect to the cross correlation coefficient threshold. As the results show, all feature sets include a substantial amount of highly correlated features. Since a lower threshold leads to more features to be removed, we choose to retain features with less than 95% correlation.

To understand the impact of various methods, we measure accuracy of attribution at each step of feature elimination. The results are presented in Figures 4, 5, 6, 7, 8. The results after removing common code with SSDEEP are referred to as *RAW*, after standardization as *STD*, after cross-correlation and variance analysis are labelled as *CC*, after

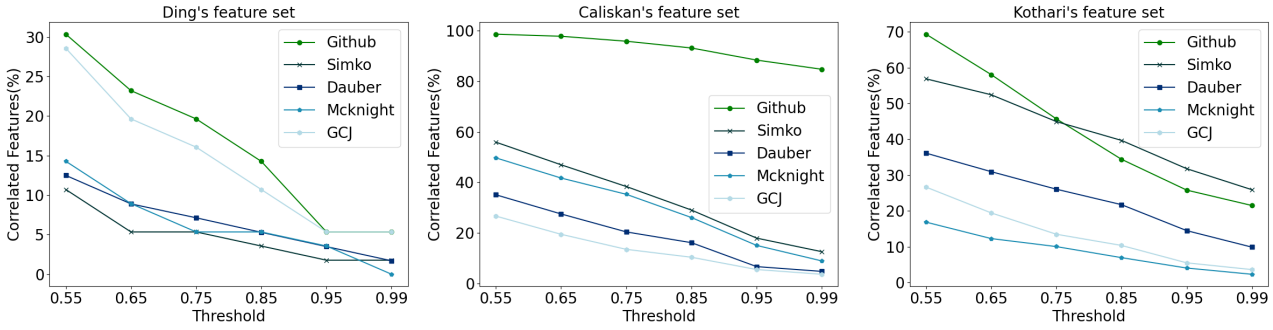



Fig. 3. Pearson Correlation Analysis on GitHub dataset

LassoLars elimination as *LL*, after PCA and Information gain approaches as *PCA* and *IG*, respectively. Due to the massive number of Caliskan's features, we could not obtain results for *RAW* and *STD* data (even with our significant computing resources).

There are several aspects that are clearly observable from the results of our attribution experiments.

**First**, somewhat expected, but the important outcome of these experiments is that removing noise, statistically correlated and unimportant features with certain methods does not significantly affect the accuracy. The accuracy of attribution remains consistent for all feature sets for most of the classifiers.

Tho feature selection methods that negatively influence accuracy are PCA and IG. LassoLars step significantly reduces the amount of features for analysis, making the data more manageable while retaining the same level of accuracy as previous steps, which is essential for attribution approaches dealing with enormous features sets such as Caliskan's features (Table 5). For example, LassoLars method reduces the amount of features retaining only 0.002% of overall features (GitHub data) for Caliskan's features. The reduction is less significant for Kothari's and Ding's features as these are initially much smaller sets.

The PCA and IG methods further reduce the number of features, yet at the expense of accuracy. For example, using Caliskan's features we can attribute a code to an author in GCJ dataset with 61%(RF) on the reduced with LassoLars method features (326 features), yet, using PCA reduction (27 features) accuracy drops to only 37% and with IG the accuracy drops even further. We have conducted additional experiments that showed that applying IG method after LassoLars approach before PCA does not result in any significant improvement in accuracy.

**Second**, among employed classifiers, Random Forest performs consistently well across all experiments. Logistic Regression, Gaussian NB and Neural Network are the second best classifiers that perform reasonably good. On the other hand, Linear Discriminant Analysis and SVC have unstable and low performance across different elimination steps.

**Third**, the feature set matters. The Ding's features produce the most stable (feature selection agnostic), yet the least accurate performance, falling to less than 20% on GitHub dataset. The Kothari's features produce the most accurate results (100% on Simko's data, 99% on GCJ data). The Caliskan's features are most susceptible to the fluctuations of feature elimination (over 20% difference in accuracy between LassoLars and PCA feature selection on McKnight and GCJ data with Random Forest).

It is also interesting to observe the effect of removing duplicated code (*RAW*) on the results. Both Ding's and Caliskan's feature sets are sensitive to the removal of common code among files. The accuracy of Ding's features dips from 63% (GCJ data) to over 35% with Simko's dataset. However, the accuracy with Kothari's features remains the same irrespective of common code removal.

**Forth**, the accuracy varies depending on the data. Both the Ding's and Caliskan's feature sets produce the lowest accuracy on GitHub dataset (20% and 5%, respectively). The highest accuracy is achieved on GCJ (almost 60% with Ding's features) and Simko set (almost 80% with Caliskan's features).

All feature sets provide consistent, yet low accuracy results (ranging from 40% to 53%) on the Dauber dataset that consists of small code snippets. Although the original study reported accuracy of 70%, we treat this data as an outlier.

**Finally**, it should be noted that we were not able to obtain the results claimed by the studies on any of the datasets.

On the one hand, our analysis was performed with data settings different from the ones reported in their original studies. On the other hand, attribution "in-the-wild" will rarely conform to the characteristics of the dataset outlined in the paper. For example, Caliskan's study was able to obtain 98% accuracy on a set of 250 authors with 9 samples per authors each ranging from 68 to 83 lines of code. None of the sets used in our analysis comply with these strict criteria.

### 4.3 Permutation Test Analysis

Permutation tests was conducted on the reduced set of features (i.e., feature set obtained after all feature elimination steps) on GitHub dataset with Random Forest classifier. The results are shown in Figure 9. The dashed line represents accuracy of the reduced feature set is 20.1% with Ding's features, 5.2% with Caliskan's features, and 58% with Kothari's features. The bars depict frequency of accuracy for runs with permuted features. A p-value was computed to estimate if the results obtained by shuffling values fall within the range

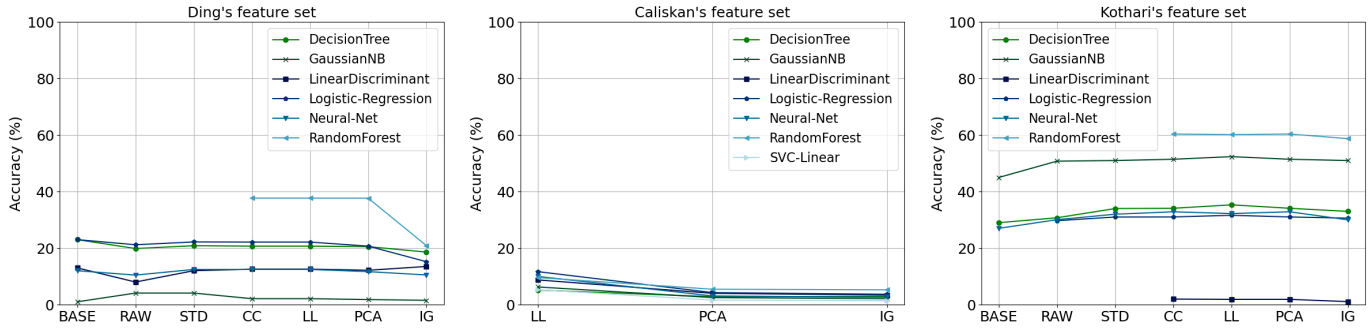



Fig. 4. Feature elimination for Github set

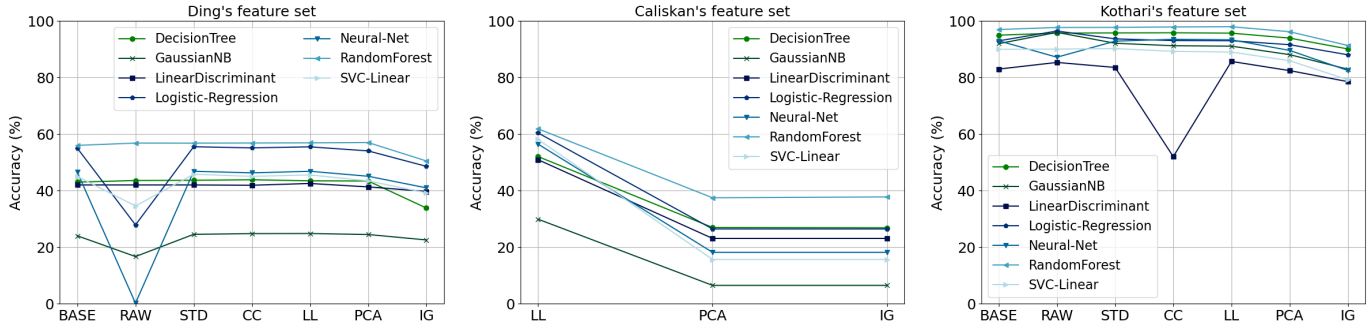



Fig. 5. Feature elimination for GCJ_Java set

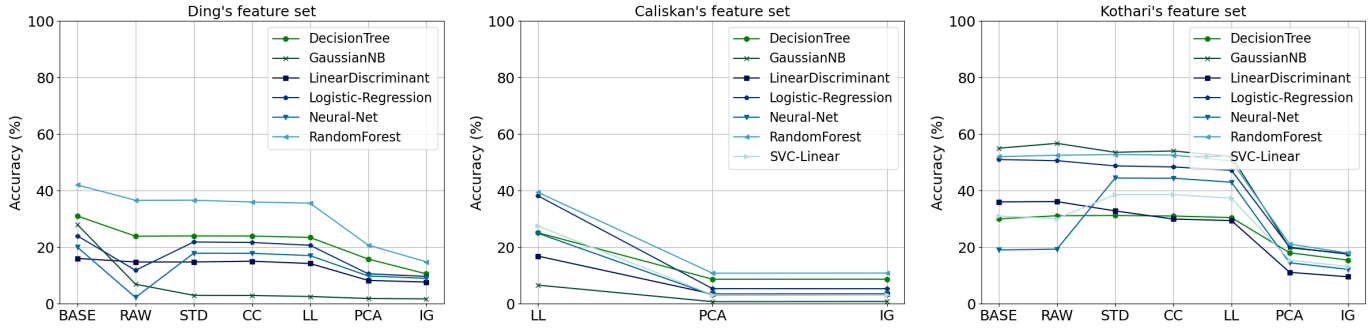



Fig. 6. Feature elimination for Mcknight set

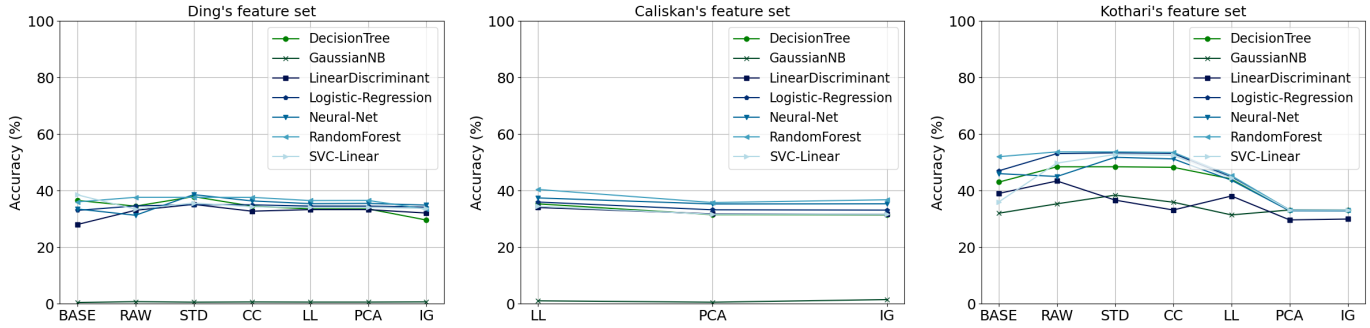



Fig. 7. Feature elimination for Dauber set

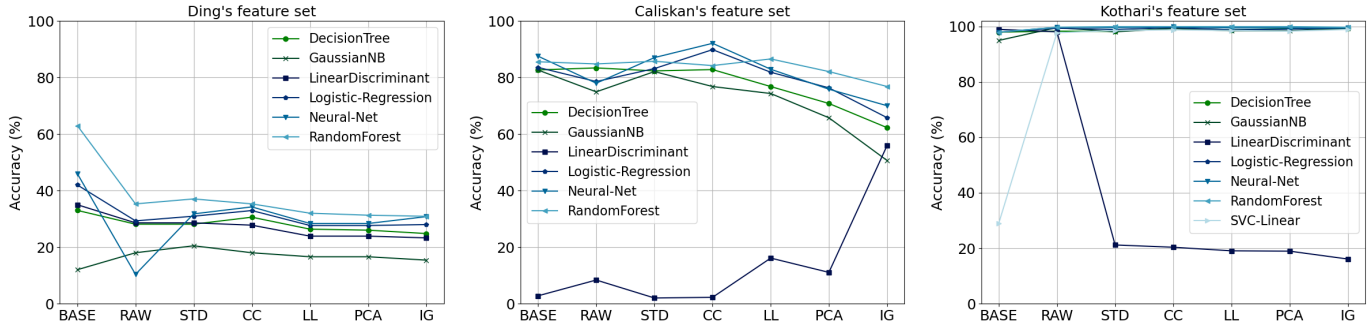



Fig. 8. Feature elimination for Simko set

TABLE 5
Percentage of removed features after feature elimination

| Features | FE Method | Removed Features (Number of Remaining Features) | | | | |
|---|---|---|---|---|---|---|
| | | Github | GCJ | McKnight | Dauber | Simko |
| Ding features | RAW | 0% (56) | 0%(56) | 0%(56) | 0% (56) | 0% (56) |
| | STD | 0% (56) | 0%(56) | 0%(56) | 0% (56) | 0% (56) |
| | CC | 5% (53) | 5%(53) | 3%(54) | 73%(15) | 1%(55) |
| | LL | 5%(53) | 8%(51) | 48%(29) | 73%(15) | 67%(18) |
| | PCA | 16%(47) | 19%(45) | 19%(45) | 73%(15) | 67%(18) |
| | IG | 39%(34) | 42%(32) | 85%(8) | 75%(14) | 73%(15) |
| Caliskan features | LL | 99.9%(706) | 99.6%(638) | 99.9%(326) | 99.9%(211) | 99.2%(279) |
| | PCA | 99.9%(21) | 99.9%(27) | 99.9%(29) | 99.9%(28) | 99.8%(62) |
| | IG | 99.9%(21) | 99.9%(27) | 99.9%(26) | 99.9%(16) | 99.9%(32) |
| Kothari features | RAW | 0%(644) | 0%(697) | 0%(2005) | 0%(1028) | 0%(494) |
| | STD | 0%(644) | 0%(697) | 0%(2005) | 0%(1028) | 0%(494) |
| | CC | 4%(617) | 11%(618) | 4%(1924) | 11%(905) | 31%(337) |
| | LL | 6%(600) | 13%(606) | 16%(1674) | 78%(217) | 64%(173) |
| | PCA | 46%(344) | 59%(280) | 94%(113) | 94%(53) | 70%(147) |
| | IG | 89%(68) | 91%(57) | 96%(64) | 96%(37) | 90%(47) |

of values that can be considered as significantly important within 99% of probability.

The accuracy of the permuted feature set is much lower than the accuracy of the reduced feature set which indicates that the reduced feature sets are statistically significant and randomizing even one of the features decreases the accuracy drastically. The reduced feature sets are available for research community [6].

## 5 Impact of Dataset Characteristics on Attribution Accuracy

Our experiments clearly show that using an existing attribution features on a different data does not reliably guarantee similar performance. The datasets employed in our experiments vary from the original studies in several aspects: the number of authors available in the dataset, the number of code samples per author and thus an overall number of files, and the number of lines of code in the available samples. We thus venture to explore the impact of each of these characteristics on the accuracy of attribution. For our experiment, we use the Github dataset with the reduced features sets (after IG step), yet, the experiments with the reduced sets produced after LassoLars method produce the same tendencies with a better accuracy. Our goal in this section is to understand the behavior of feature sets in different dataset settings.

6. https://cyberlab.usask.ca/authorattribution.html

### 5.1 How many authors are needed?

Traditionally, author attribution techniques have been applied to problems where a small set of candidate authors is known in advance (e.g., plagiarism detection). As the pool of authors increases, this task becomes more difficult. Individual author style has to be unique enough to consistently distinguish itself from a large number of other potential authors.

In previous research, the smallest number of authors (2) was used by Pellin et al. [31]. The majority of the studies in the field generally have up to 20 authors [6], [15], [33], [23], [35], [21], [33], [16], [17], [24], [34]. The higher number of authors is less common, i.e, 29 - 70 authors were used in [14], [32], [15], [36], [37], [19], [20], [22], 106 authors in [26] and 525 authors in [25].

The large-scale authorship attribution analysis was explored only by Caliskan et al. [18] and Abuhamad et al. [38]. Caliskan et al. analysis reached 98% accuracy with a 250 author set, and 93% accuracy with 1,600 authors. However, a follow-up study by Simko et al. [20] that mirrored [18] methodology on the same dataset for 5, 20, and 50 authors received a different result - 100%, 88.2%, and 84.5%, respectively. Abuhamad et al. analysis with 8,903 authors achieved 92%.

In this experiment, we explore the impact of a varying number of authors. It should be noted that authors and samples have an inverse relationship in our Github data, i.e., a large number of authors only have a few code samples. To maintain enough author variation for the experiments, we selected the largest number of samples that exists in our Github data with the smallest number of authors (10) and the largest number of authors. As a result, in this experimental round we use 9 samples that are, on average, 110 lines long. With 9 samples per author we can have up to 2000 authors. We randomly select 10, 30, 250, 750, 1000, 1500 and 2000 authors from the Github dataset. All reported results are averaged cross-validated values obtained from 4 runs with random selections of authors.

As it is illustrated in Figure 10, accuracy decreases dramatically by increasing number of authors up until 250 authors. After that the accuracy decrease is less noticeable. Increasing number of authors consequently increases a number of features, as a result, for example, we cannot attribute authors using Caliskan features with more than 750 authors (a dataset with 2000 authors and 9 sample per

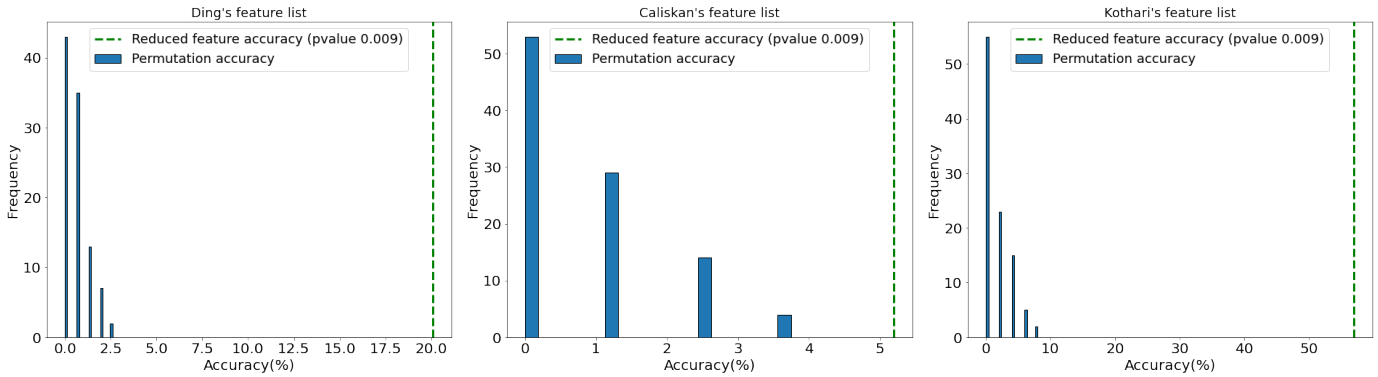



Fig. 9. Permutation test analysis

author produces 1,225,774 Caliskan's features, in comparison, the same set has 646 Kothari's features).

**Results:** As the pool of authors increases, authorship attribution becomes more difficult. Individual author style has to be unique enough to consistently distinguish itself from a large number of other candidate authors. The results show that attributing a code among 10 authors is straightforward when using the Random Forest classifier with the Kothari or Caliskan's features. Generally, Kothari features discriminate authors better than other feature sets.

### 5.2 How long should code samples be?

The length of code samples employed in previous studies ranged significantly. For example, Burrows et al. [35] used programs varying from 1 to 10,789 lines of code, Tennyson et al. [17] took samples ranging from 1 to 3265 lines, Caliskan et al. [18] worked with samples that fluctuated in size from 68 to 83 lines of code. In forensics and malware analysis, often only small source code snippets may be available, thus understanding the impact of code length on attribution accuracy is critical.

Whether these results were coincidental or a direct result of a smaller number of lines in considered code samples is not clear. Our analysis showed that Dauber's dataset is highly redundant (Figure 2) and not equally distributed, i.e., 3,125 programs were one-line samples, 2,171 had from 2 to 9 lines, 445 programs had from 10-99 lines, and 34 samples had from 100 to 554 lines.

For a fine-grained view, we group our data by code size into the following subsets: *10 to 50*, *51 to 100*, *101 to 250*, *251 to 500*, *501 to 750*, *751 to 1000*, *1001 to 1500*, *1501 to 2000*, and *2001 to 4000* LOC. Since the number of lines in code varies, we fix the number of samples per author to 5 and number of authors to 15 for each collection to have enough data that can support higher ranges of lines of code (2001,4000).

**Results:** As it is illustrated in Figure 11, increasing number of lines of code has a different impact on attribution accuracy. There is no detectable increase or decrease trend on the graphs for neither of the feature sets. Kothari's features produce stable and high accuracy. Both Caliskan's and Ding's features have drastically variable performance. It is should noted that in spite of this variability Ding's features achieve lower performance with short code samples (10-50 LOC) and the highest accuracy with the longest in our dataset files (4000 LOC).

### 5.3 How many samples per author are needed?

Since authorship attribution aims to derive a developer's coding style, having sufficient data for this analysis becomes essential. The number of samples used in attribution domain across the years varied significantly, from 3 [34] and 9 samples [18] to 28-128 [23], and 1,360 - 7,900 samples per author [31]. In most cases, the selection of programs per author is driven by data availability. Thus, the question of "how many samples per author are needed for analysis" remains. We look into this question through the lens of the size of samples. Following the practise of our previous experiments, we randomly select 5, 10, 20, 50, 100, 250, 750, and 1000 samples from Github dataset. For this experiment, we fixed the number of authors to 15 with samples with [1-500] LOC.

**Results** Attributing a small number of samples is challenging for all feature sets. Yet after a certain number of samples is reached the accuracy plateaus for all feature sets (Figure 12). Kothari's features produce stable accuracy with more than 10 samples per author, Caliskan's features with more than 20 samples per author, and Ding's features with more than 250 code samples per author. This is an important finding since availability of code samples typically presents challenges in attribution research.

TABLE 6
The recommended dataset characteristics

| Feature set | Ding's features | Caliskan's features | Kothari's features |
|---|---|---|---|
| Num. Authors | 10 | 10 | 10 |
| LOC | [251-500], [2001-4000] | [251-500] | [251-1000] |
| Num. Samples/ Author | 250+ | 20+ | 10+ |

### 5.4 Recommended data characteristics

Table 6 gives a summary of data characteristics that are most likely to produce the best accuracy results for the three feature sets. It is interesting to note that some characteristics are common across all sets, such as 10 authors with samples containing [251-500] lines of code. If available, then 250 samples per author for all feature sets will achieve the best accuracy.

In order to verify the recommended parameters, we generate five subsets of our datasets. Since the Dauber and

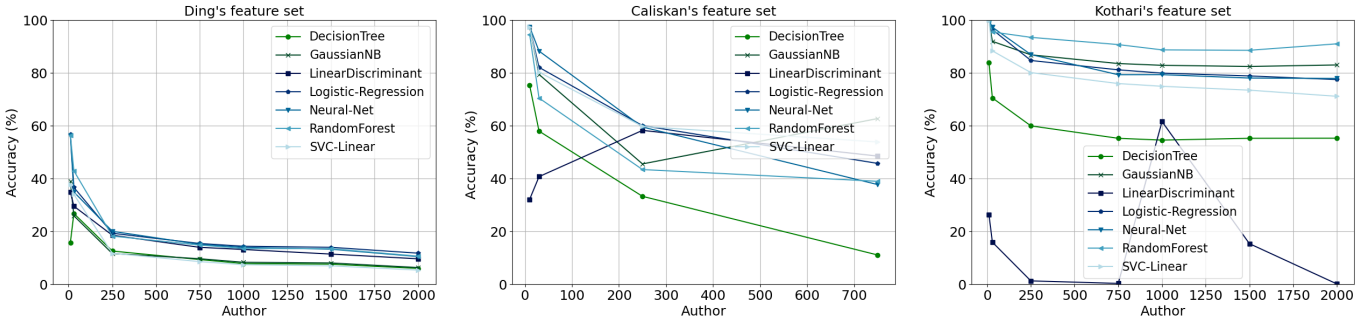



Fig. 10. Attribution accuracy for varying number of authors

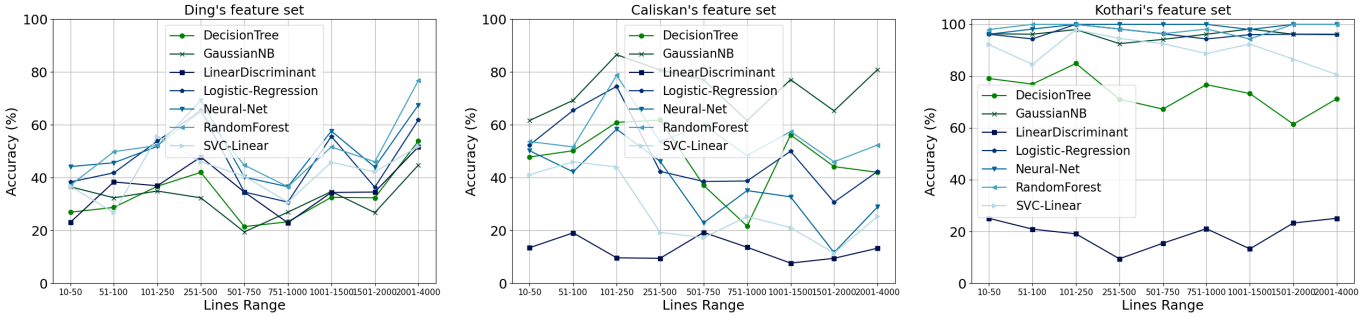



Fig. 11. Attribution accuracy for varying number of lines of code

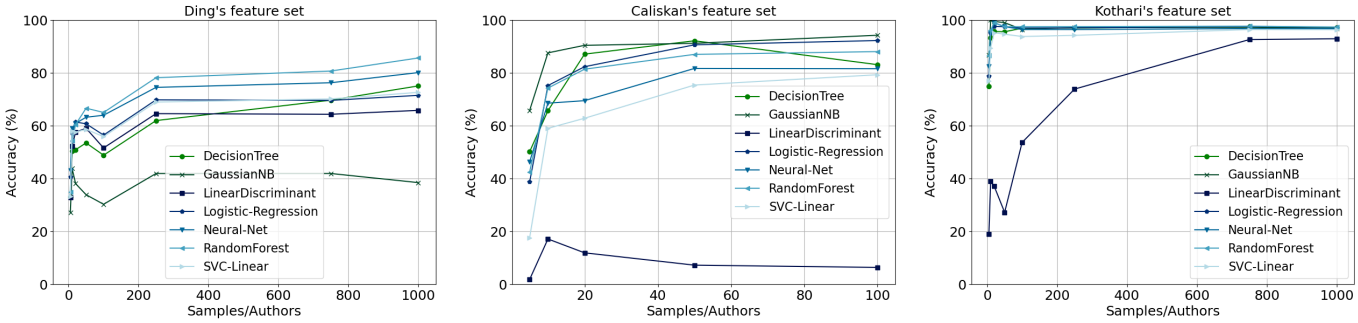



Fig. 12. Attribution accuracy of varying number of samples per author

Simko datasets have lower LOC in average, we adjust LOC ranges to the closest available range. Similarly, since 250 samples per author are not available for all datasets, we use 20 code samples per author with an understanding that this is not the optimal characteristic when using the Ding's features. For this experiment, we use the proposed feature elimination approach stopping after the LassoLars selection to avoid a decrease in accuracy that we encountered after PCA and IG.

The results are presented in Table 7. The obtained attribution accuracy is significantly higher compared to our baseline results or the results obtained during the feature elimination experiments.

## 6 Conclusion

Finding a proper dataset for an authorship attribution study is a challenging task. The researchers often resort to use an available dataset with all its shortcomings. How these shortcomings affect the results of attribution is not always clear.

In this work, we investigated how the quality of dataset impacts the overall performance of authorship attribution. This is the first systematic study that offers a practical insight into the performance of the existing attribution studies and looks into the potential bias that different datasets' shortcomings may bring. From the series of experiments, we reached several important conclusions.

**As a number of authors increases, attribution becomes more challenging**. Our experiments show that it is straightforward to attribute a small number of authors for any feature set (100% or nearly perfect accuracy using the Caliskan and the Kothari features). However, increasing the number of authors results in a visible deterioration in performance. The most significant decrease in accuracy happens with the

TABLE 7
Accuracy with the recommended dataset characteristics

| | Dataset Characteristics | | | Ding's features | | | | Caliskan's features | | | | Kothari's features | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | #Auth. | #Sample/ Author | LOC | RF | GNB | LR | NN | RF | GNB | LR | NN | RF | GNB | LR | NN |
| Github | 10 | 20 | [251-500] | 55% | 44% | 56% | **58**% | 82% | 73% | **84**% | **84**% | 96% | **100**% | 96% | 97% |
| GCJ | 10 | 20 | [251-500] | **98**% | 90% | 95% | 96% | **100**% | 97% | 99% | 99% | **100**% | 99% | **100**% | **100**% |
| McKnight | 10 | 20 | [251-500] | 62% | 40% | 48% | **54**% | 77% | 68% | **80**% | **80**% | 88% | **95**% | 91% | 93% |
| Dauber | 10 | 20 | [200-500] | 47% | 38% | **51**% | **51**% | 84% | 72% | 86% | **90**% | 90% | **96**% | 91% | 93% |
| Simko | 10 | 20 | [20-502] | **49**% | 36% | 48% | 45% | **88**% | 77% | **88**% | **88**% | **100**% | 97% | 98% | 96% |

Ding's and the Caliskan's features. Thus it is important to understand that an accuracy of attribution will likely decrease when a large number of authors is present, and an analysis of a new attribution method should explore a range of authors.

**Large code samples do not translate to better accuracy**. It is equally challenging to accurately attribute small code snippets (10 to 50 LOC) as it is to attribute longer files (e.g., with 1500-2000 LOC). This is an important finding as having an option of using smaller code samples when data is scarce, is beneficial.

**Attribution with over 250 samples per author does not result in a corresponding increase in accuracy**. Attributing a small number of samples is challenging, yet once a critical mass of samples per author is offered, accuracy plateaus.

In a nutshell, part of the results obtained by previous studies were due to coincidental match between data and approach, or due to the attribution technique's tolerance to dataset imperfections. If appropriate characteristics for dataset are not selected properly, result are not reliable and cannot be valid for other datasets.

## REFERENCES


[1] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with jplag," *J. UCS*, vol. 8, no. 11, p. 1016, 2002.

[2] J.-H. Ji, G. Woo, and H.-G. Cho, "A plagiarism detection technique for java program using bytecode analysis," in *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, vol. 1. IEEE, 2008, pp. 1092–1098.

[3] J. L. Donaldson, A.-M. Lancaster, and P. H. Sposato, "A plagiarism detection system," in *ACM SIGCSE Bulletin*, vol. 13, no. 1. ACM, 1981, pp. 21–25.

[4] M. L. Gavrilova and R. Yampolskiy, "Applying biometric principles to avatar recognition," in *Transactions on computational science XII*. Springer, 2011, pp. 140–158.

[5] E. H. Spafford and S. A. Weeber, "Software forensics: Can we track code to its authors?" *Computers & Security*, vol. 12, no. 6, pp. 585–595, 1993.

[6] S. G. MacDonell, A. R. Gray, G. MacLennan, and P. J. Sallis, "Software forensics for discriminating between program authors using case-based reasoning, feed forward neural networks and multiple discriminant analysis," in *Proceedings of the 6th International Conference on Neural Information Processing (ICONIP'99)*, vol. 1. IEEE, 1999, pp. 66–71.

[7] F. Ullah, J. Wang, S. Jabbar, F. Al-Turjman, and M. Alazab, "Source code authorship attribution using hybrid approach of program dependence graph and deep learning model," *IEEE Access*, vol. 7, pp. 141 987–141 999, 2019.

[8] S. Afroz, A. C. Islam, A. Stolerman, R. Greenstadt, and D. McCoy, "Doppelgänger finder: Taking stylometry to the underground," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. USA: IEEE Computer Society, 2014, p. 212226.

[9] S. Alrabaee, P. Shirani, M. Debbabi, and L. Wang, "On the feasibility of malware authorship attribution," in *International Symposium on Foundations and Practice of Security*. Springer, 2016, pp. 256–272.

[10] R. Chouchane, N. Stakhanova, A. Walenstein, and A. Lakhotia, "Detecting machine-morphed malware variants via engine attribution," *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 3, pp. 137–157, 2013.

[11] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Authorship attribution of android apps," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY 18. New York, NY, USA: Association for Computing Machinery, 2018, p. 277286.

[12] V. Kalgutkar, N. Stakhanova, P. Cook, and A. Matyukhina, "Android authorship attribution through string analysis," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. New York, NY, USA: Association for Computing Machinery, 2018.

[13] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, and A. Matyukhina, "Code authorship attribution: Methods and challenges," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–36, 2019.

[14] I. Krsul and E. H. Spafford, "Authorship analysis: Identifying the author of a program," *Computers & Security*, vol. 16, no. 3, pp. 233–257, 1997.

[15] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas, "Source code author identification based on n-gram author profiles," *Artificial Intelligence Applications and Innovations*, pp. 508–515, 2006.

[16] B. S. Elenbogen and N. Seliya, "Detecting outsourced student programming assignments," *Journal of Computing Sciences in Colleges*, vol. 23, no. 3, pp. 50–57, 2008.

[17] M. F. Tennyson and F. J. Mitropoulos, "A bayesian ensemble classifier for source code authorship attribution," in *International Conference on Similarity Search and Applications*. Springer, 2014, pp. 265–276.

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

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

[20] L. Simko, L. Zettlemoyer, and T. Kohno, "Recognizing and imitating programmer style: Adversaries in program authorship attribution," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 1, pp. 127–144, 2018.

[21] M. Gull, T. Zia, and M. Ilyas, "Source code author attribution using authors programming style and code smells," *International Journal of Intelligent Systems and Applications*, 2017.

[22] C. Zhang, S. Wang, J. Wu, and Z. Niu, "Authorship identification of source codes," in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*. Springer, 2017, pp. 282–296.

[23] U. Bandara and G. Wijayarathna, "Source code author identification with unsupervised feature learning," *Pattern Recognition Letters*, vol. 34, no. 3, pp. 330–334, 2013.

[24] M. Shevertalov, J. Kothari, E. Stehle, and S. Mancoridis, "On the use of discretized source code metrics for author identification," in *Search Based Software Engineering, 2009 1st International Symposium on*. IEEE, 2009, pp. 69–78.

[25] C. McKnight, "Stylecounsel: Seeing the (random) forest for the trees in adversarial code stylometry," Master's thesis, University of Waterloo, 2018.

[26] E. Dauber, A. Caliskan-Islam, R. Harang, and R. Greenstadt, "Git

blame who?: Stylistic authorship attribution of small, incomplete source code fragments," *arXiv preprint arXiv:1701.05681*, 2017.

[27] S. Burrows, A. L. Uitdenbogerd, and A. Turpin, "Comparing techniques for authorship attribution of source code," *Software: Practice and Experience*, vol. 44, no. 1, pp. 1–32, 2014.

[28] B. Stein, S. M. zu Eissen, and M. Potthast, "Strategies for retrieving plagiarized documents," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 825–826.

[29] X. Meng and B. P. Miller, "Binary code multi-author identification in multi-toolchain scenarios," 2018.

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

[31] B. N. Pellin, "Using classification techniques to determine source code authorship," *Department of Computer Science, University of Wisconsin*, 2000.

[32] H. Ding and M. H. Samadzadeh, "Extraction of java program fingerprints for software authorship identification," *Journal of Systems and Software*, vol. 72, no. 1, pp. 49–57, 2004.

[33] J. Kothari, M. Shevertalov, E. Stehle, and S. Mancoridis, "A probabilistic approach to source code authorship identification," in *Information Technology, 2007. ITNG'07. Fourth International Conference on*. IEEE, 2007, pp. 243–248.

[34] R. C. Lange and S. Mancoridis, "Using code metric histograms and genetic algorithms to perform author identification for software forensics," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 2082–2089.

[35] S. Burrows, A. L. Uitdenbogerd, and A. Turpin, "Application of information retrieval techniques for source code authorship attribution," in *International Conference on Database Systems for Advanced Applications*. Springer, 2009, pp. 699–713.

[36] W. Wisse and C. Veenman, "Scripting dna: Identifying the javascript programmer," *Digital Investigation*, vol. 15, pp. 61–71, 2015.

[37] X. Yang, G. Xu, Q. Li, Y. Guo, and M. Zhang, "Authorship attribution of source code by using back propagation neural network based on particle swarm optimization," *PloS one*, vol. 12, no. 11, p. e0187204, 2017.

[38] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang, "Large-scale and language-oblivious code authorship identification," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS 18. New York, NY, USA: Association for Computing Machinery, 2018, p. 101114.

[39] F. Ullah, J. Wang, S. Jabbar, F. Al-Turjman, and M. Alazab, "Source code authorship attribution using hybrid approach of program dependence graph and deep learning model," *IEEE Access*, vol. 7, pp. 141 987–141 999, 2019.

[40] G. Frantzeskou, S. Gritzalis, and S. G. MacDonell, "Source code authorship analysis for supporting the cybercrime investigation process," *Handbook of Research on Computational Forensics, Digital Crime, and Investigation: Methods and Solutions*, pp. 470–495, 2004.

[41] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas, "Effective identification of source code authors using byte-level information," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 893–896.

[42] G. Frantzeskou, E. Stamatatos, S. Gritzalis, C. E. Chaski, and B. S. Howald, "Identifying authorship by byte-level n-grams: The source code author profile (scap) method," *International Journal of Digital Evidence*, vol. 6, no. 1, pp. 1–18, 2007.

[43] G. Frantzeskou, S. MacDonell, E. Stamatatos, and S. Gritzalis, "Examining the significance of high-level programming features in source code author classification," *Journal of Systems and Software*, vol. 81, no. 3, pp. 447–460, 2008.

[44] G. Frantzeskou, E. Stamatatos, and S. Gritzalis, "Supporting the cybercrime investigation process: effective discrimination of source code authors based on byte-level information," in *International Conference on E-Business and Telecommunication Networks*. Springer, 2005, pp. 163–173.

[45] G. Frantzeskou, "The source code author profile (scap) method: An empirical software engineering approach," Ph.D. dissertation, PhD thesis, Department of Information and Communication Systems, University of the Aegean, Mytilene, Greece, 2007.

[46] E. Quiring, A. Maier, and K. Rieck, "Misleading authorship attribution of source code using adversarial learning," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 479–496.

[47] A. Matyukhina, N. Stakhanova, M. Dalla Preda, and C. Perley, "Adversarial authorship attribution in open-source projects," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY 19. New York, NY, USA: Association for Computing Machinery, 2019, p. 291302.

[48] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital Investigation*, vol. 3, pp. 91 – 97, 2006, the Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[49] A. Gray, S. MacDonell, and P. Sallis, "Software forensics: Extending authorship analysis techniques to computer programs," *The Journal of Law and Information Science*, 1997.

[50] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[52] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human faces," *Josa a*, vol. 4, no. 3, pp. 519–524, 1987.

[53] T. Bayes, "Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s," *Philosophical transactions of the Royal Society of London*, no. 53, pp. 370–418, 1763.

[54] M. Aly, "Survey on multiclass classification methods," *Neural Netw*, vol. 19, pp. 1–9, 2005.

[55] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[56] L. Breiman, F. Jerome, S. Charles J., and O. R.A., *Classification and regression trees*. Wadsworth International Group, 1984.

[57] P. A. Lachenbruch and M. Goldstein, "Discriminant analysis," *Biometrics*, pp. 69–85, 1979.

[58] V. Vapnik, "Pattern recognition using generalized portrait method," *Automation and remote control*, vol. 24, pp. 774–780, 1963.

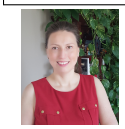
[59] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[60] D. R. Cox and E. J. Snell, *Analysis of binary data*. CRC press, 1989, vol. 32.

[61] A. S. Arefin, R. Vimieiro, C. Riveros, H. Craig, and P. Moscato, "An information theoretic clustering approach for unveiling authorship affinities in shakespearean era plays and poems," *PloS one*, vol. 9, no. 10, p. e111445, 2014.

PLACE PHOTO HERE

PLACE PHOTO HERE



**Farzaneh Abazari** is a Postdoctoral fellow in the Department of Computer Science at the University of Saskatchewan, Canada. She received her Ph.D. in 2018 in Computer Engineering. Her research interests include authorship attribution, network security and software analysis.

**Enrico Branca** is a researcher in the Department of Computer Science at the University of Saskatchewan, Canada. He has been working in information security for over a decade with experience in software security, information security management, and cyber security R&D.

**Norah Ridley** is a bachelor Student in the Department of Computer Science at University of Saskatchewan. Her main research interests are machine learning and security.

**Natalia Stakhanova** is the Canada Research Chair in Security and Privacy, and an Associate Professor at the University of Saskatchewan, Canada.

**Mila Dalla Preda** is an Associate Professor at the Department of Computer Science at the University of Veron, Italy. Her main research interest are formal methods applied to software protection, software security, program and malware analysis.