Zero-Shot Source Code Author Identification: A Lexicon and Layout Independent Approach

Pegah Hozhabrierdi*
Department of EECS
Syracuse University
Syracuse, USA
phozhabr@syr.edu

Dunai Fuentes Hitos*
Independent Researcher
Huelva, Spain
dunai.fuenteshitos@alumni.epfl.ch

Chilukuri K. Mohan
Department of EECS
Syracuse University
Syracuse, USA
mohan@syr.edu

***Abstract*—We tackle the challenge of *Zero-Shot* identification of authors of source code, which can be used with no prior samples of authors outside of the training data. In our approach, a feedforward neural network is first trained on a multi-class classification task. Then, a substantial part of this network is duplicated and reused to compare code samples. We refer to this design as *Feedforward Duplicated Resolver (FDR)* model. We propose new input features to train this model, called *Variable-Independent Nested Bigrams*, extracted from the Abstract Syntax Trees of code samples. These features provide robustness against lexical and layout obfuscation attacks frequently used in plagiarism attempts. This approach performs accurately even on code samples from unknown authors, on data obtained from *Google Code Jam*, an international coding competition platform. For example, for the task of predicting whether a pair of samples from 43 unknown authors have been written by the same person, we obtain an AUC of 0.96 and 0.91 for non-obfuscated and obfuscated code, respectively.**

***Index Terms*—Author identification, Source code stylometry, Zero-shot learning, Obfuscation**

I. Introduction

Source code author identification attempts to discover the true authors of source code samples, and is important in many challenging applications such as detecting classroom plagiarism, copyright violations, and malware authorship attribution [1], [3], [13], [24].

Current source code author identification models rely heavily on a rich set of code samples from each author for training [4], [6], [7], [11]. However, in a real-world setting, scarcity of the historical record of the authors' code samples makes such training impossible. Additionally, to support identification of yet unseen authors, model retraining is needed.

We address an important question unanswered by most existing studies: *How can we identify unknown authors with no available prior code samples?* Many real-world scenarios fall under this category. For example, consider the classroom plagiarism detection task for programming assignments, with very few or no prior samples per student that could be used for training a classifier. To address this problem, many plagiarism tools in educational institutions use lexical and/or layout similarities between the submitted assignments [19]. These similarities, mostly borrowed from text authorship attribution, can be obfuscated easily. Lexical and layout features (examples shown in Table I) can be manipulated by simple changes in variable names, spacing, and ordering of the statements without altering the syntax. Hence there is a need for a classifier that (1) requires no prior samples (zero-shot classification); and (2) is robust against lexical and layout attacks (obfuscation independence) [26].

TABLE I: Source code stylometric features

Features	Examples
Lexical	Number of occurrences of keywords Number of unigrams Number of comments Number of strings, character, and numeric literals
Layout	Number of tab characters Number of space characters Number of empty lines Number of whitespace characters
Syntactic	Nested Bigrams Maximum depth of an AST node Term frequency AST node bigrams Term frequency of code unigrams in AST leaves
Dynamic	Number of function calls Average running time per module Memory usage of each function Memory access patterns

Zero-shot learning is important for scenarios in which:

1) Acquiring prior code samples written by an author is an infeasible task if their identity is unknown.
2) The number and/or length of the samples may be insufficient for learning.

This paper presents the first work that successfully implements zero-shot learning of coding style, integrating a zero-shot solution and a similarity measure that help protect against layout and lexical obfuscations. The model used here is based on duplicating a trained feedforward network, extensible to authors with no prior samples in the training or feature extraction phase. We will refer to this model as *Feedforward Duplicated Resolver (FDR)* throughout the paper. To vectorize code samples, we use two different feature sets. First is the *Nested Bigrams* (NB) representation, proposed in our previous work [11]. To provide robustness against lexical obfuscation attacks, we introduce a lexicon-independent variation,

[*] Equal contribution

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

Variable-Independent Nested Bigrams (VINB), as our second feature set. These features are extracted from the Abstract Syntax Tree (AST) of the code, and capture the coding style without the need for additional features.

The language selected for this study is Python, due to its growing popularity and the paucity of plagiarism studies for it. However, our approach can be applied to other languages and embeddings as well. On a dataset of 246 authors with 8 Python code samples per author from Google Code Jam, we obtained AUC results of 0.96 and 0.91 for Nested Bigrams and Variable Independent Nested Bigrams respectively.[1]

In Section II, we discuss related work. Section III describes the feature sets we use. Section IV presents our neural network architecture and training approach. Experimental setup and results are discussed in Section V. The last section presents conclusions and future work.

II. Related Work

Prior work (in [2], [5], [6], [11], [14], [20], [22]) has addressed authorship attribution as a classification task. Given a set of authors, each code sample must be attributed to one member of this restricted set. Models are trained to discriminate between these classes, a "closed-world" task. These models are not useful for identifying the authorship of a code sample outside the set of known authors or the training distribution, an "open-world" task. Comparing code samples from previously unseen authors and determining their stylistic similarity is an important and practical real-world problem that has received little attention so far.

Two existing works are of interest in the context of the problem we address:

Caliskan et al. [6]: The existence of an unknown author is considered in the context of a binary class discrimination problem. This is achieved by setting a threshold on the number of consenting trees in a random forest classifier. If the set of the majority vote has less trees than the threshold, the code is labeled as "unknown". Their classifier cannot capture the similarity of different samples written by the same unknown author. For the addition of new unknown authors, the classifier has to be retrained.

Wang et al. [25]: They generalize the classifier to unseen authors by using Siamese matching networks [16]. Their matching classifier, SUNDAE, uses both dynamic and static features, examples of which are shown in Table I. This study suffers from the following shortcomings:

- **Choice of evaluation metric:** Using the overall accuracy to evaluate a classifier is misleading when the data set is imbalanced. Indeed, their accuracy of 99.91% for a dataset of 1,159 authors, with 229 code samples per author, can be achieved merely by categorizing every data point as belonging to the majority class (i.e. "no match"); the number of pairs of code samples belonging to the same author is $\binom{229}{2} \times 1,159$ whereas the total number of pairs of code samples is $\binom{1,159\times 229}{2}$, and the ratio is $< 0.1\%$. In such cases, it is necessary to evaluate the classifier using diagnostic test assessments such as negative and positive predictive values (NPV and PPV), F1 score, and AUC.
- **Execution-dependence:** The extraction of dynamic features require executing code, which can be costly and risky (e.g., due to malicious code fragments).
- **Necessity of dynamic features:** Performance is much poorer when restricted to static features.
- **Slow Training:** For a set of n code samples, the Siamese network is to be trained with $\binom{n}{2}$ pairs of code samples. The required time for training SUNDAE grows quadratically as a function of its sample size.
- **Slow Testing:** Comparing a new code sample against a large database, even with precomputed vector representations, implies applying a trained binary classifier for each pair to predict the likelihood of a match.

III. Feature Set

Since the early studies on source code stylometric analysis, finding the right feature set has remained a major challenge. In our previous work [11], we introduced new syntactic features, *Nested Bigrams (NB)*, and showed their efficacy in capturing a Python programmer's coding style while being robust against layout obfuscation. This section describes Nested Bigrams and then introduces *Variable-Independent Nested Bigrams*, new features that are resilient against both layout and lexical obfuscation attempts.

A. Nested Bigrams

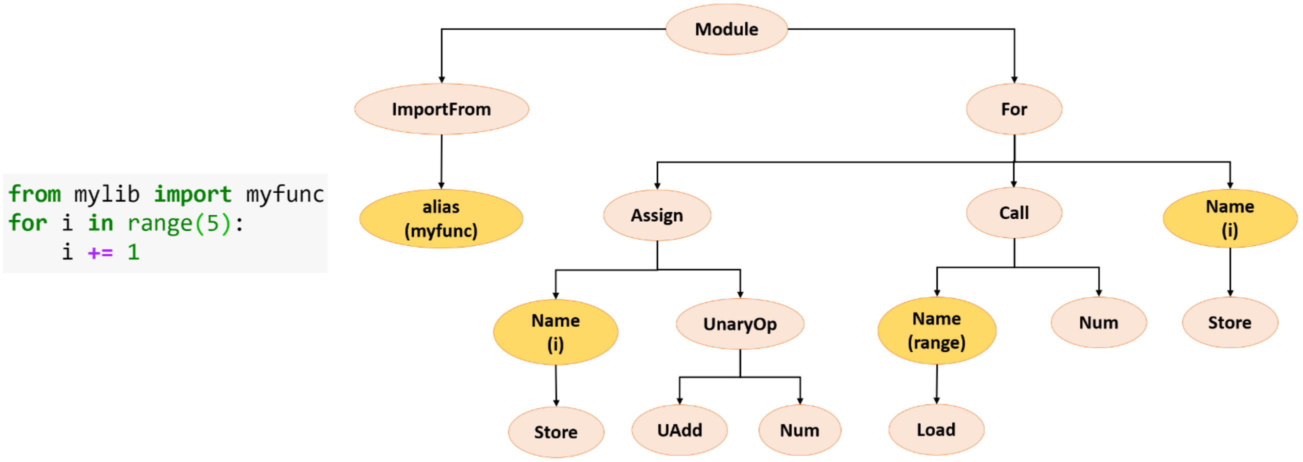
Nested Bigrams represent parent-child node pairs in the abstract syntax tree, where each node carries the information of all its descendants. They use nesting of subtrees to preserve the information between non-adjacent nodes, including sibling nodes and descendants, in the abstract syntax tree (AST).

For example, in Figure 1, the left hand side shows a simple code snippet (in Python) whose AST is depicted on the right. Nodes `Assign` and `Call` are sibling nodes, and `UAdd` is a non-immediate descendent of `Assign`. The parent-child pair (`Assign` , `UnaryOp`) is a bigram. The corresponding Nested Bigram includes the information concerning subtrees (and all descendants) of nodes `Assign` and `UnaryOp` respectively, i.e. (`Assign(Name(Store),UnaryOp(UAdd, Num))`, `UnaryOp(UAdd, Num)`).

Nested Bigrams embedding vectorizes the source code using the frequencies of bigrams in the code. When the embedding is applied to two code samples from the same author, the results are expected to be similar; but if the authors are different, the embedding maps them to substantially dissimilar vectors. For each AST, we omit the non-informative root node *Module* in the feature extraction step.

Nested Bigrams are both fast to extract and accurate in representing the coding style of an author, maintaining information about the long-distance connections in the AST without the use of complicated encoding mechanisms [4]. The

[1] Our model and implementation details are available at https://github.com/Pegayus/zeroshot-code-authorship.



```
from mylib import myfunc
for i in range(5):
    i += 1
```

Fig. 1: A Python code snippet (left) and its abstract syntax tree (right); (ImportFrom, alias(myfunc)) is an example of a Nested Bigram. Nodes highlighted in yellow contain the names of variable/functions, and are replaced by a default keyword in VINB.

performance of Nested Bigrams is compared with prior state-of-the-art features in [11].

B. Variable-Independent Nested Bigrams

Layout and lexical obfuscation techniques are among the most widely used methods for obfuscating code [26]. Nested Bigrams are sensitive to the lexical information in the names chosen for the variables and functions. The dependency of Nested Bigrams on lexical features degrades their performance in the presence of such obfuscation attacks (Figure 6b). To solve this problem, we replace the variable and function names by a default keyword (such as 'VAR') in the abstract syntax tree. We refer to the Nested Bigrams of this modified syntax tree as *Variable-Independent Nested Bigrams (VINB)*. In Figure 1, the nodes containing layout information are highlighted (yellow). In a variable-independent AST, the values of these nodes will be replaced by a default keyword before the extraction of Nested Bigrams.

In the case of non-obfuscated code, a slight drop of performance can be expected for VINB when compared to Nested Bigrams. The choices of specific names in a non-obfuscated script can reflect the personal taste and style of a programmer, exhibited repeatedly in much of their source code. Despite losing this valuable information, VINB offers competitive performance for unseen authors and with obfuscated code.

IV. Feedforward Duplicated Resolver (FDR) Neural Network Model

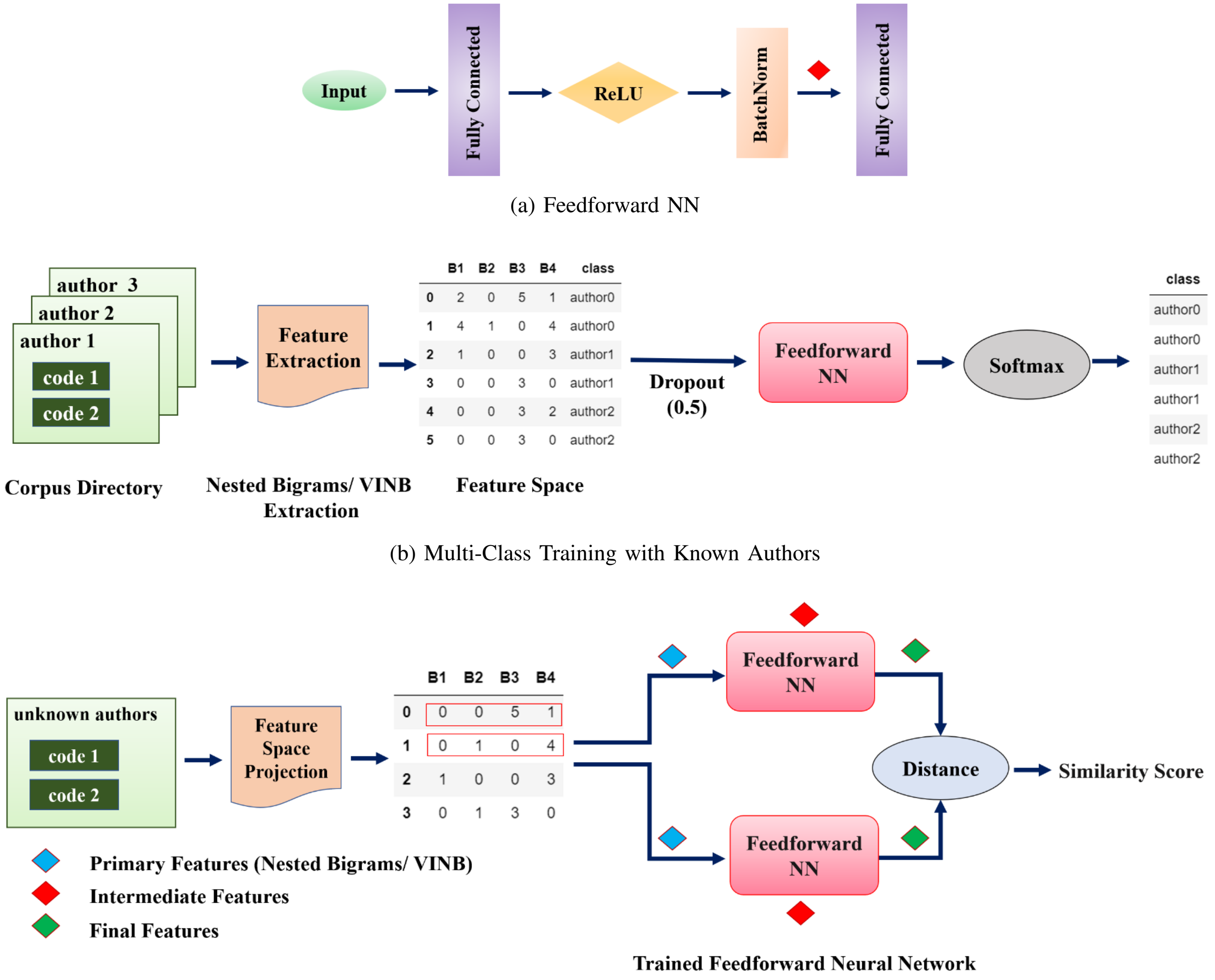
Siamese networks have been used for one-shot and zero-shot learning tasks in image recognition and short text classification [16], [27]. A Siamese network uses two identical networks to generate two output vectors from a pair of samples. These two vectors are then compared to determine whether they belong to the same class. Siamese networks are trained by matching pairs (or triplets [10]) of samples from available data. Instead of a class label, they receive a binary label.

Training Siamese networks is often slow and inefficient. Recently, it has been demonstrated [21] that a SoftMax loss on a multi-class classification problem is equivalent to a smoothed triplet loss. Hence, instead of using Siamese Networks, we propose to use SoftMax loss minimization. During inference, as in the Siamese case, the network is duplicated to compare samples. We refer to this new approach as the *Feedforward Duplicated Resolver (FDR)*, consisting of the following components:

1) A **feedforward** multi-class discrimination neural network is first trained, with its inputs being the NB/VINB feature vectors, as shown in Figure 2b. The output layer of this network applies a *SoftMax* function to the outputs of the final layer of "hidden nodes" in the network. The training objective is to minimize the categorical cross-entropy loss. Our experiments use a feedforward network with two layers, with Dropout [23], ReLU activation, and Batch Normalization [12]. The implementation is in PyTorch [8]. Training is done with the Adam [15] optimizer using a learning rate schedule with warm restarts [17]. We do not apply any data augmentation techniques, which highlights the data efficiency of this approach.
2) The **duplicated** part of the system consists of two copies of the above trained feedforward module without the SoftMax layer.
3) The **resolver** compares the outputs of the above (final) feature vectors, using input (NB/VINB) vectors from two code samples, that may be of unknown origin, including authors whose code has not been previously encountered. We measure the cosine similarity between different output vectors for the final fully connected layer (Figure 2). Training the feedforward network to minimize the categorical cross-entropy maximizes the cosine similarity of samples of the same class.

Baselines. We compare different methods to construct code embeddings:

- *Nested Bigram Normalized Frequencies:* Vectorization of the samples into bigram frequencies provides a powerful



(a) Feedforward NN

	B1	B2	B3	B4	class
0	2	0	5	1	author0
1	4	1	0	4	author0
2	1	0	0	3	author1
3	0	0	3	0	author1
4	0	0	3	2	author2
5	0	0	3	0	author2

class
author0
author0
author1
author1
author2
author2

(b) Multi-Class Training with Known Authors

	B1	B2	B3	B4
0	0	0	5	1
1	0	1	0	4
2	1	0	0	3
3	0	1	3	0

(c) Testing with Unknown Authors

Fig. 2: *FDR Approach:* **(a) Feedforward neural network** used as backbone in (b) and (c). **(b) Multi-class training** of the feedforward network: The feature extraction block obtains the frequencies of the most frequent bigrams ($B_1, B_2, ..., B_i$) among all samples, normalized by sample's length. The samples are further encoded into the corresponding feature space and used to train the network. The numbers shown beneath each layer of the feedforward network denote the output dimensions. **(c) Testing** on unseen authors: The feature projection block maps each sample into the feature space obtained above. The three colored diamonds identify the three different embeddings used in experiments; the exact location of the red diamond is shown in (a).

TABLE II: Evaluation of different embedding methods in the 81-class classification task, using a k Nearest Neighbor classifier (with k =5). Accuracy is reported for unseen code samples from authors within the training distribution.

Method	Output Dimension	Metric	Driving Force	Accuracy (%)
Random Guess	NA	NA	NA	1.23
Nested Bigrams	7,269	Cosine	NA	55.55
PCA	300	Cosine	Maximum Variance	54.32
Siamese Network	5	Euclidean	Contrastive Loss [9]	60.49
SUNDAE-NB	500	NA	Binary Cross-Entropy Loss	40.74
Feedforward NN	81	Cosine	Cross-Entropy Loss	**95.06**

embedding.

- *Principal Component Analysis (PCA)*: This method can be used to reduce the dimension of Nested Bigram embedding.
- *Siamese Network:* We train a Siamese network [28] to minimize intra-class and maximize inter-class distance. The final fully connected layer has as many units as the embedding space dimensionality (instead of the number

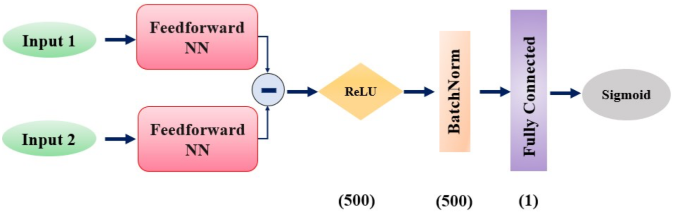


Fig. 3: SUNDAE-NB architecture; the number below each layer denotes the output dimension of the layer.

of target classes).

- *SUNDAE-NB:* This network uses NB/VINB feature sets along with a neural network classifier built on top of a Siamese network [25], as shown in Figure 3.
- *Feedforward NN Hidden Layer Outputs:* We use intermediate outputs of a neural network classifier, i.e., outputs of the last fully connected layer in Figure 2a.

Similarity Metrics. Similarity between samples in the embedding space can be computed in many ways. We have experimented with (1) spatial proximity, as given by the Euclidean Distance, and (2) direction alignment, as given by the Cosine Similarity.

Dataset. *Google Code Jam (GCJ)* is an annual international programming competition that has been in operation since 2003. With a publicly available code repository on a variety of algorithmic problems, GCJ has been used in previous studies on code de-anonymization ([4], [6], [11]). We use this repository due to its popularity in the field and the richness of data. To build the corpus, we crawled seven years of competition submissions (2006 - 2014) in Python, and sorted them by the number of submissions per user. The experiments reported in this section were performed on a dataset of 81 users with 9 problems each. This is a smaller subset of data than used for the experimental results reported in Section V.

Experiments. To evaluate our approach, we compared different embeddings and similarity metrics. Tests were conducted in the first fold of stratified cross-validation, ensuring that each class label appears during training. To compute within-training-distribution performance, we first train the embedding model and then train a 5-Nearest Neighbor classifier using the pairwise distance matrix of the resulting embeddings. We evaluate its accuracy on newly embedded inputs from a test set of code samples from the same authors. We have optimized the output dimension and other hyper-parameters for each baseline, when applicable (hence the different output dimension in Table II).

Results. As shown in Table II, the feedforward neural network approach outperforms others by a considerable margin. Reducing the dimensionality of Nested Bigrams using PCA from over 7000 to 300 resulted in little change in the performance. Further dimensionality reduction with PCA, however, affected the performance negatively. A Siamese network trained with Contrastive Loss [9], the best performing configuration found with Siamese training, could achieve dimensionality reduction to a number as low as 5.

V. Zero-Shot Author Identification

Our goal is to identify the similarity of any pair of code samples using the FDR network. The backbone is the feedforward neural network, trained for multi-class classification of a large set of code samples from known authors. After training the backbone, we duplicate it and use it as a resolver for testing, measuring the cosine similarity between the outputs.

A. Experimental Setup

Dataset. In the following experiments, we use a dataset of 246 authors with 8 code samples each from GCJ. We choose this larger dataset (compared to that in section IV) to demonstrate an important behavior discussed below and shown in Figure 5. We use up to 160 authors for training, 43 for validation, and the remaining 43 authors for testing.

Training. We consider the learning task on a dataset of n_{train} labeled source code samples belonging to a_{train} authors. To keep the training unbiased towards an author's style, we let code samples be uniformly distributed among authors, i.e., each author has c_{train} number of code samples and $n_{train} = a_{train} \times c_{train}$. The feature extraction block extracts the most frequent NB/VINB features from training data. We restrict the number of bigrams, taking only the most frequent ones that are used by at least two authors. This helps to avoid overfitting and aims to establish a general feature space. Taking only the most frequent nested bigrams also helps establish a linear growth rate for our dataset size with increasing number of samples (Figure 4). Then, the processed data is used to train our feedforward neural network on the supervised task of author classification.

The effect of varying the number of nested bigrams and the number of authors (on the validation AUC) can be seen in Figure 5. We find that the top-8000 most frequent nested bigrams constitute a good feature space (the same results were obtained for VINB). This validation has been done for different values of x in top-x features (from 500 to 10,000 over intervals of 500), omitted from Figure 5 for clarity. The number of all bigrams for each number of training authors is annotated on the *All Frequent* curve in Figure 5.

Testing Outside of Distribution. To test the performance of the model on *unknown authors* (i.e., with no samples in the training set), we construct a mixed set of $n_{test} = a_{test} \times c_{test}$ source code samples that belong to authors outside of the training distribution. The first step is to map these samples into the feature space of the training set. No new bigrams are added to the feature set in this stage. We build the Resolver network by adopting the trained feedforward neural network module, and estimating the similarity of each pair, e.g., using Cosine Similarity, defined as:

$$K(e_1, e_2) = \frac{< e_1, e_2 >}{||e_1|| \cdot ||e_2||}$$

where e_i is the bigram embedding of the i^{th} source code. The results are reported for $(a_{train}, c_{train}) = (160, 8)$ and $(a_{val}, c_{val}) = (43, 8)$ for selecting the optimal number of nested bigrams, and $(a_{test}, c_{test}) = (43, 8)$.

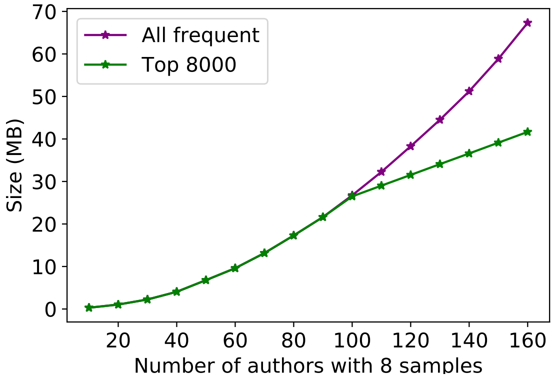


Fig. 4: Memory requirement for Nested Bigrams/VINB. *All frequent* includes all bigrams that have been used by at least two authors. The size of this feature set grows exponentially with respect to number of authors. Restricting the set to the top 8000 bigrams eliminates this effect.

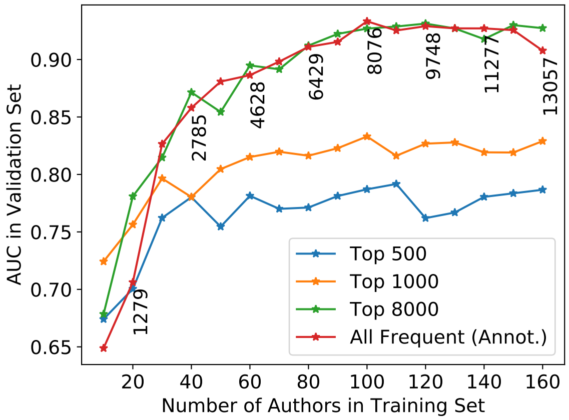
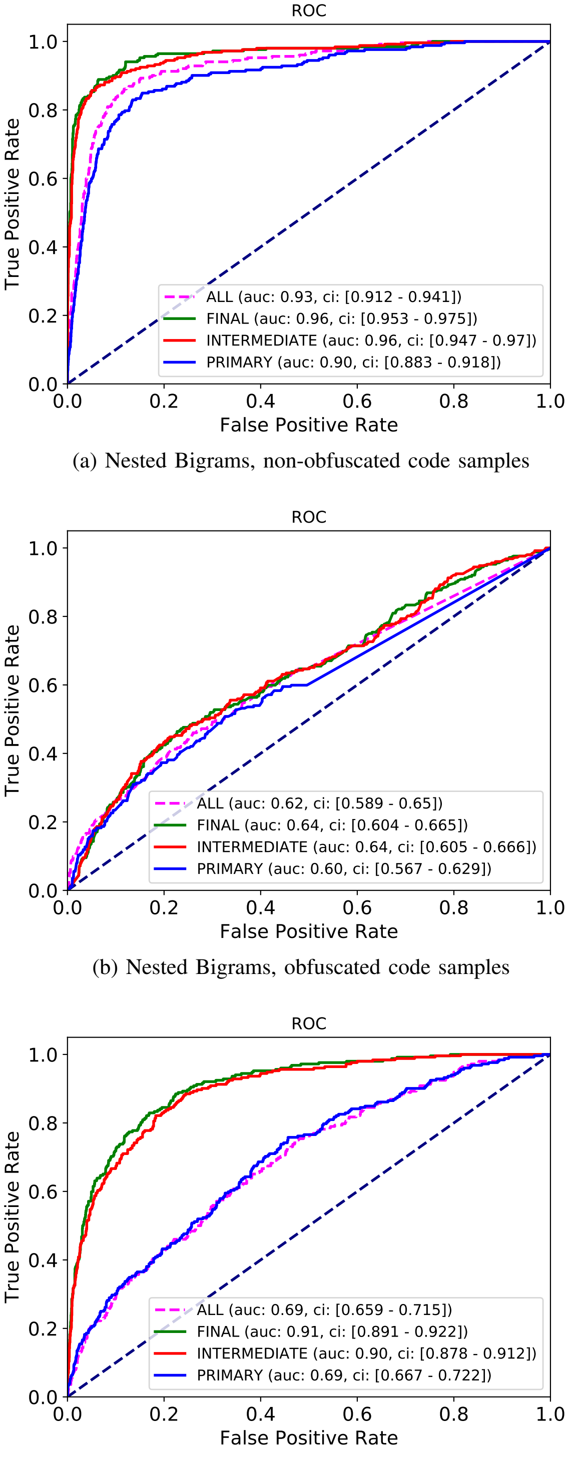


Fig. 5: AUC results for different numbers of nested bigrams and authors. Each curve corresponds to a specific limit to the number of nested bigrams considered, with the red curve corresponding to no limit. The red curve is annotated by the number of bigrams.

B. Results

Using the three embeddings extracted from the locations indicated using shaded diamonds in Figure 2c, the ROCs are shown in Figure 6 for both Nested Bigrams and VINB. The bigram embeddings (PRIMARY features) are both weaker than the FINAL embeddings generated by the proposed FDR network model. The best AUC values for both bigrams are achieved by the FINAL features embedding, with a 5-point reduction in AUC for VINB, a trade-off for being robust to lexical obfuscation.

Discussion. A drawback of selecting only “the most frequent” bigrams across authors (in the training set) is the elimination of the unique properties of an author that cannot be generalized to others (e.g., the specific choices of names for variables and class structures). We show this effect by comparing the performance of “the most frequent” or PRIMARY

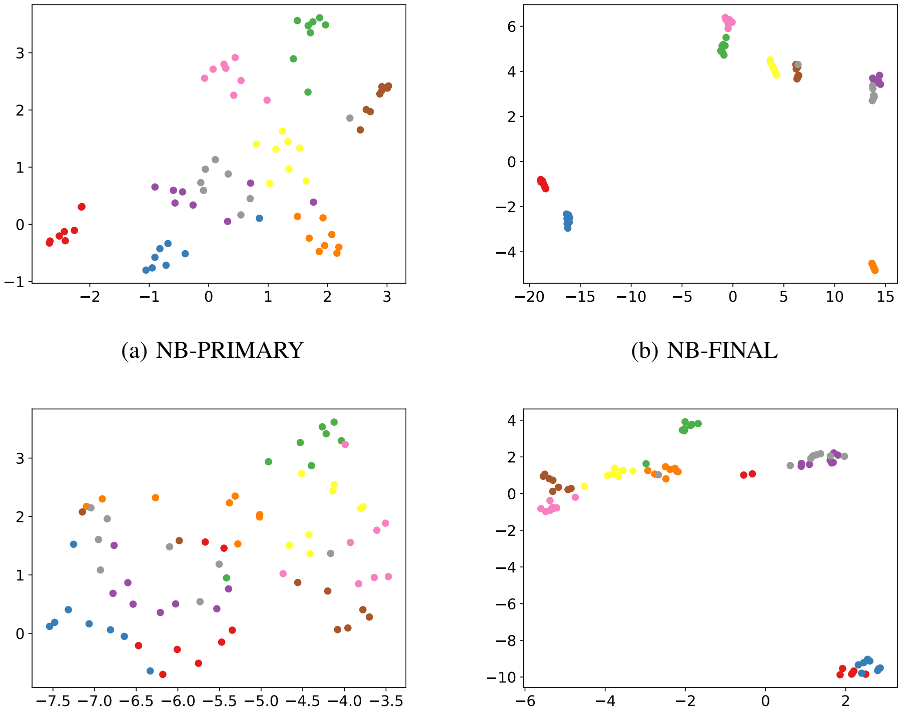


(a) Nested Bigrams, non-obfuscated code samples

(b) Nested Bigrams, obfuscated code samples

(c) VINB, obfuscated code samples

Fig. 6: ROC for zero-shot learning using (a) Nested Bigrams, (b) Nested Bigrams extracted from obfuscated code and (c) Variable-Independent Nested Bigrams (VINB) extracted from obfuscated code. FINAL, INTERMEDIATE, and PRIMARY features refer to the embeddings indicated in Figure 2c. ALL feature includes all Nested Bigrams/VINB extracted from the set of samples with unknown authors; and *ci* is the 90 percent confidence interval of the reported AUC.



(a) NB-PRIMARY

(b) NB-FINAL

(c) VINB-PRIMARY

(d) VINB-FINAL

Fig. 7: UMAP 2D visualizations of embeddings computed for 72 code samples from 9 unseen authors, 8 code each. Samples with the same authorship are displayed with the same color. For PRIMARY, the embedding space is defined by the Nested Bigrams / VINB of a disjoint set of 203 authors. For FINAL, projections to the embedding space were learned from the same disjoint set of authors. We use the same UMAP hyperparameter for all plots, namely: 10 neighbors and 0 minimum distance.

features to that of all bigrams (not limited to those in the training set) or ALL features. The ROC for these features is shown in the dashed magenta curve. Since these features include all bigrams of each source code sample, they are 10 times larger in size than the PRIMARY feature space.

The results surpass the PRIMARY features for zero-shot recognition, but only when variable names are considered. In general, the ROC curves for Nested Bigrams show a slightly better discriminative ability than those for VINB. This hints at the importance of user-defined variable names in characterizing one's coding style. However, this small drop in VINB performance in zero-shot recognition can be an acceptable trade-off for immunity against lexical obfuscation attacks. For a more intuitive understanding, we present a visualization in Figure 7 of a 2D UMAP [18] with the projections of the code samples for the first 9 authors from our test set.

Plagiarism Detection. Challenges confronted (in the classroom scenario) include the lack of prior coding samples from students, and the possibility that the layout and lexical features of the code are changed by plagiarizers. With our proposed FDR solution, the coding assignments of a class can be evaluated without any prior information on new students' coding habits. Only a training set obtained from previous students in the course is needed. This is due to our model's capability of identifying the similarity of two source code samples written by the same author, without having prior samples from this author in the database.

VI. Conclusion

We introduced the challenge of zero-shot recognition for source code author attribution. We showed that among many possible embeddings for source code, including the previous studies, the internal representations of feedforward neural networks trained on multi-class classification yields the highest inside-distribution accuracy (95 percent) when used along with the k-Nearest Neighbor approach. The implementation of this network as a Resolver for outside-distribution testing demonstrates a strong discriminative capability for unseen authors (with the AUC of 0.96 for 43 unknown authors). We also presented a new code vectorization method resilient to lexical and layout obfuscations, Variable Independent Nested Bigrams (VINB), and showed competitive performance with the same setup. Our contributions are as follows:

1) We introduced the Zero-Shot Code Authorship Identification challenge on a dataset of 1,968 source code belonging to 246 authors from Google Code Jam.
2) We proposed a Feedforward Duplicated Resolver network model to answer the zero-shot recognition chal-

lenge. This model is trained in a multi-class classification task, using a feedforward neural network and a SoftMax. We showed that this training scheme is faster and more accurate than training of Siamese networks on binary (match/no-match) labels.

3) We introduced a modification of the Nested Bigrams representation, i.e., Variable Independent Nested Bigrams (VINB), shown to be robust against lexical and layout obfuscation attacks.
4) Our zero-shot plagiarism detection package is open-sourced for Python source code and can be used in practical settings, such as classrooms.

Limitations. This study has focused on Python source code de-anonymization on the Google Code Jam dataset; data from other application contexts remain to be explored. A possible scenario in which our approach can miss the classification of unknown authors is where the length of the provided sample code is too small to capture a specific coding behavior. An interesting study would be the analysis of optimal minimum code length for which satisfactory performance is achieved. Finally, we note that Nested Bigrams and VINBs are appropriate when sufficient *nestedness* is observed in the code samples; when there is no nesting behavior in the code samples, the Nested Bigrams will be equivalent to term frequency of node bigrams (in ASTs).

Future Work. We plan to apply this approach to code from other languages such as Ruby. Other directions of future work include applying the proposed network model using additional sets of features, including dynamic features, where available (e.g., from sandbox testing). We would also like to explore a clustering-based network model such as LVQ, instead of the feedforward network module, which may help when the same author's coding style may follow more than one pattern. Adding a multiple-authorship detection method on top of the current model may add more insight to the final results. Finding features that are globally resilient to more complex obfuscation attacks also remains a challenge. Finally, it will be of interest to analyze the performance of our FDR model on a larger set of code samples.

References

[1] Cheryl L Aasheim, Paige S Rutner, Lixin Li, and Susan R Williams. Plagiarism and programming: A survey of student attitudes. *Journal of information systems education*, 23(3):297–313, 2012.

[2] Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. N-gram-based detection of new malicious code. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 41–42. IEEE, 2004.

[3] Saed Alrabaee, Paria Shirani, Mourad Debbabi, and Lingyu Wang. On the feasibility of malware authorship attribution. In *International Symposium on Foundations and Practice of Security*, pages 256–272. Springer, 2016.

[4] Bander Alsulami, Edwin Dauber, Richard Harang, Spiros Mancoridis, and Rachel Greenstadt. Source code authorship attribution using long short-term memory based networks. In *European Symposium on Research in Computer Security*, pages 65–82. Springer, 2017.

[5] Steven Burrows and Seyed MM Tahaghoghi. Source code authorship attribution using n-grams. In *Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University*, pages 32–39. Citeseer, 2007.

[6] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security), Washington, DC*, 2015.

[7] Edwin Dauber, Aylin Caliskan, Richard Harang, Gregory Shearer, Michael Weisman, Frederica Nelson, and Rachel Greenstadt. Git blame who?: Stylistic authorship attribution of small, incomplete source code fragments. *Proceedings on Privacy Enhancing Technologies*, 2019(3):389–408, 2019.

[8] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[9] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *null*, pages 1735–1742. IEEE, 2006.

[10] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[11] Pegah Hozhabrierdi, Dunai Fuentes Hitos, and Chilukuri K Mohan. Python source code de-anonymization using nested bigrams. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 23–28. IEEE, 2018.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Mike Joy, Georgina Cosma, Jane Yin-Kim Yau, and Jane Sinclair. Source code plagiarism—a student perspective. *IEEE Transactions on Education*, 54(1):125–132, 2010.

[14] RI Kilgour, AR Gray, PJ Sallis, and SG MacDonell. A fuzzy logic approach to computer software source code authorship analysis. 1998.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[17] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[19] Mohamed El Bachir Menai and Nailah Salah Al-Hassoun. Similarity detection in java programming assignments. In *2010 5th International Conference on Computer Science & Education*, pages 356–361. IEEE, 2010.

[20] Paul W Oman and Curtis R Cook. Programming style authorship analysis. In *Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pages 320–326. ACM, 1989.

[21] Qi Qian, Lei Shang, Baigui Sun, Juhua Hu, Hao Li, and Rong Jin. Softtriple loss: Deep metric learning without triplet sampling. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6450–6458, 2019.

[22] Eugene H Spafford and Stephen A Weeber. Software forensics: Can we track code to its authors? 1992.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] Dewi Tresnawati, Arief Syaichu, et al. Plagiarism detection system design for programming assignment in virtual classroom based on moodle. *Procedia-Social and Behavioral Sciences*, 67:114–122, 2012.

[25] Ningfei Wang, Shouling Ji, and Ting Wang. 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. ACM, 2018.

[26] Hui Xu, Yangfan Zhou, Yu Kang, and Michael R Lyu. On secure and usable program obfuscation: A survey. *arXiv preprint arXiv:1710.01139*, 2017.

[27] Leiming Yan, Yuhui Zheng, and Jie Cao. Few-shot learning for short text classification. *Multimedia Tools and Applications*, 77(22):29799–29810, 2018.

[28] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2017.