

# Source Code Authorship Attribution using Stacked classifier

Chanchal Suman<sup>a</sup>, Ayush Raj<sup>b</sup>, Sriparna Saha<sup>a</sup> and Pushpak Bhattacharyya<sup>a</sup>

<sup>a</sup>Indian Institute of Technology Patna, India

<sup>c</sup>IIT Bhubaneswar, India

## Abstract

Source code authorship attribution is a process of identifying the author of a given code. With increasing number of software submissions on open source repositories like Github, Codalab, Kaggle, Codeforces online judge, etc. the authors can copy other's code for their products. The application area of this method includes the detection of plagiarized code and prevent legal issues. In this work, we have applied the tf-idf based mechanism for representing the given source code. Word and character n-grams are used for the generation of code vectors. Finally, the generated vectors are fed to different available machine learning classifiers for the prediction task. We have applied this methodology to the dataset released by organizers of the AI-SOCO, a shared task of FIRE-2020. The problem statement for the task is "Given the predefined set of source code and their authors, the task is to build a system to determine which one of these authors wrote a given unseen before source code." An accuracy of 82.95% is achieved on the test data, which attained 10th position in the competition.

## Keywords

Authorship attribution, abstract syntax tree, tf-idf, stacking

## 1. Introduction

Nowadays, the identification of the author of a piece of code has become very crucial in many cases. Some of the cases are authorship dispute, malware attacks, logic bomb, proof of authorship, frauds, etc. [1, 2, 3]. Source code is very formal and restrictive in comparison to the natural languages used in our daily life. But still, there is a large degree of flexibility while writing a program [3]. The traditional way of dealing with this task is to divide the problem into two parts: i) extracting the representation vector for the author's style from the given code, and ii) building a classifier for the prediction task using the representation vector. In this work, an abstract syntax tree (AST), has been used for the tokenization of the code. tf-idf is applied to word and character n-grams for generating the code representation. Different available machine learning classifiers are used for classification purposes. Finally, the best performance is achieved using word bigrams on a stacked model. The stacked model is an ensemble of extra tree classifier, random forest classifier, and XG-Boost classifier. The Authorship Identification of

---


*Forum for Information Retrieval Evaluation 2020, December 16-20, 2020, Hyderabad, India*

EMAIL: 1821cs11@iitp.ac.in (C. Suman); b518015@iiit-bh.ac.in (A. Raj); sriparna@iitp.ac.in (S. Saha); pb@iitp.ac.in (P. Bhattacharyya)

URL: <https://github.com/chanchalIITP> (C. Suman); <https://github.com/satushsinha> (A. Raj)



© 2020 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Source Code (AI-SOCO) is the shared task organized by FIRE-2020 [4]. We participated in this task, and the accuracy achieved for the test set is 82.95%.

## **2. Concepts Used**

In this section, we have discussed the basic concepts of abstract syntax tree, and the stacked classifier.

### **2.1. Abstract Syntax Tree**

An abstract syntax tree (AST) is a representation of a program's code in the form of a rooted tree. Nodes of an AST correspond to different code constructs (e.g., math operations and variable declarations). Children of a node correspond to smaller constructs that comprise its corresponding code. Different constructs are represented with different node types [5].

### **2.2. Stacked Classifier**

Stacked Generalization is an ensemble machine learning algorithm, where different machine learning algorithms are ensembled together to get the results. Stacked Model uses a meta-learning algorithm to learn how to combine the predictions from two or more machine learning algorithms. The benefit of stacking is that better results are obtained by combining the outputs of multiple classifiers.

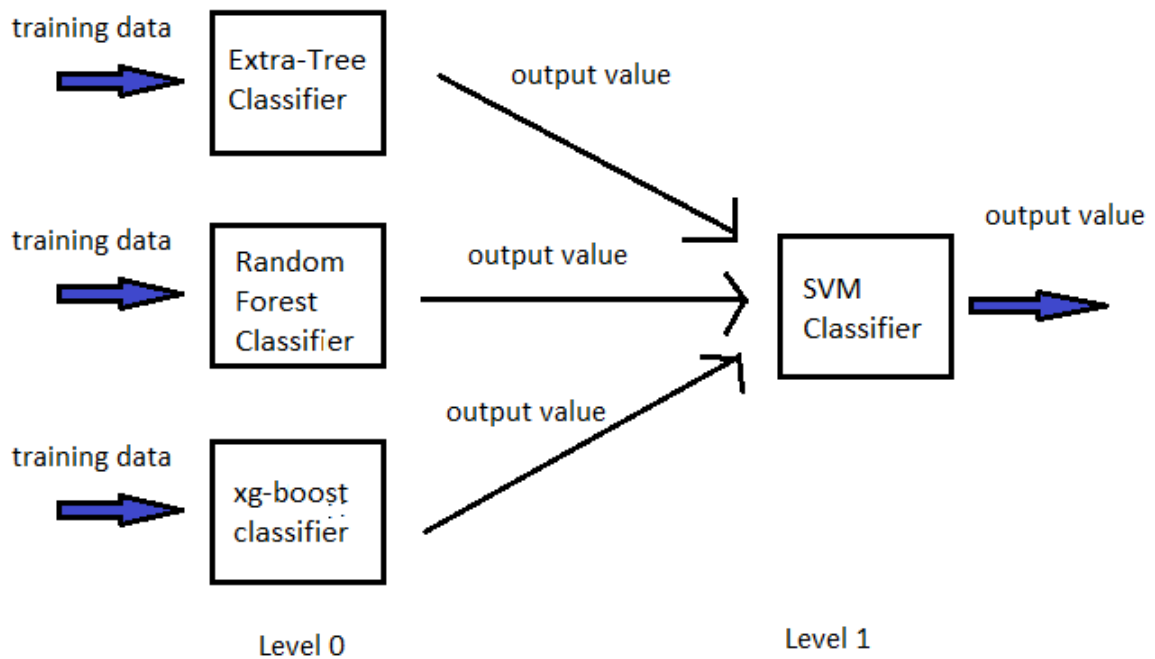
In stacking, different base machine learning classifiers are combined together with the help of a meta-classifier. The individual classification models are trained based on the complete training set. Then the outputs for these classifiers are fed as inputs to the meta-classifier where it predicts the final result. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble.

## **3. Methodology Used**

In this section, we have discussed the data, its pre-processing, vectorization, and the classification.

### **3.1. Data**

The dataset is composed of source code collected from the open submissions in the Codeforces online judge. There are 100 source code of 1,000 users, a total of 100,000 codes. The data is divided into three parts, training, development, and test set having 50000, 25000, and 25000 codes of C++ programming language, respectively. All codes are correct, bug-free, compile-ready, and for a unique problem.



**Figure 1:** The Proposed Stacked classifier

### 3.2. Data Preprocessing

The data is preprocessed using the clang library for the generation of Abstract syntax tree for the source code. The writing style and syntax of code are different from the natural texts, thus the normal tokenization process should not be usual for splitting codes. We have used AST for the above purpose. Firstly, the language of the source code is found through the GuessLang library of python. All the codes are written in C or C++, thus we have used the clang library to generate the abstract syntax tree. AST traversal provides the list of tokens for the codes.

### 3.3. Vectorization

tf-idf Vectorizer is used for generating the vector representation of the given source code. The list of tokens are extracted from the AST. Word and character n-grams are generated from the token list. While building the vocabulary, we have ignored the terms having document frequency strictly higher than 0.6 and strictly lower than 0.0. In this way, the code vectors are calculated using the traditional tf-idf mechanism. We have used word uni-grams, bi-grams, tri-grams, and character bi-grams, tri-grams, and quad-grams for the experimental purpose.

**Table 1**

Accuracy on Dev. data using word unigrams

Classifier	Accuracy (in %)
DT	69.47
RF	75.1
SVM	71.22
Adaboost	70.34
Stackd (RF+DT+Adaboost)	73
Majority voting (RF+DT+Adaboost)	74.38

### 3.4. Classifier

The representation vectors for the given codes are fed to different available machine learning classifiers. We have also used a stacked model for classification purpose. A stacking of three different classifiers i) extra tree classifier, ii) random forest, and iii) XG-Boost Classifier is carried out. The input is first classified using the three different classifiers, and then the outputs produced by the classifiers are fed to SVM for final prediction. Thus, SVM is used as the meta classifier over these classifiers. This architecture is shown in Figure 1. 150 estimators are used with the extra tree classifier and the random forest classifier. SVM classifier takes the output of all the three classifiers and then predicts the author for the input given. Stacking Classifier from the mlxtend library is used to stack the models.

## 4. Results

In this section, we have discussed the results achieved on the development data, using different approaches. An accuracy of 75.1% is achieved by random forest, using the word uni-gram based vectors. The performance for the unigrams is shown in Table 1. The accuracy value has increased with the use of uni-gram and bi-grams, shown in Table 2. An accuracy of 83.20% is achieved using the stacked model for the word n-grams ( $n=1, 2$ ). A decrease in performance is recorded with the vectors created from the combinations of word uni-grams, bi-grams, and tri-grams, as shown in Table 3. Character n-grams are also used for generating the feature representations. Word-based features outperform the character-based features. The experimental results are shown in Table 4. Thus, it is concluded that the word bi-gram-based representation with stacked model performs the best on the development dataset.

Accuracy values of 81.58% and 82.95% are achieved using the bi-grams-based representations by random forest and the stacked model, respectively. The results for the test data are shown in table 5.

## 5. Conclusion

Source code authorship identification is a process of identifying, who wrote a piece of code, given some set of probable authors. In the current work, we have used a tf-idf mechanism for generating the feature vector for the codes. The abstract syntax tree is used for tokenizing the

**Table 2**

Accuracy on Dev. data using word n-grams (n=1,2)

Classifier	Accuracy (in %)
DT	79.51
RF	82.35
SVM	75.69
Stacked Model	83.20

**Table 3**

Accuracy on Dev. data using word n-grams (n=1,2,3)

Classifier	Accuracy (in %)
DT	69.4
RF	73.45
SVM	74.72

**Table 4**

Accuracy on Dev. data using character n-grams

Classifier	Accuracy (in %)
DT (n=2)	61.18
RF (n=2)	66.27
SVM (n=2)	60.12
RF (n=3)	65.73
RF (n=4)	65.27

**Table 5**

Accuracy on test data using word bigrams

Classifier	Accuracy (in %)
RF	81.58
Stacked Model	82.95

codes and splitting it into tokens. Experimental results illustrate that word bi-grams perform better than other combinations of word and character n-grams.

## References

- [1] R. Layton, P. Watters, R. Dazeley, Automatically determining phishing campaigns using the uscap methodology, in: 2010 eCrime Researchers Summit, IEEE, 2010, pp. 1–8.
- [2] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, R. Greenstadt, De-anonymizing programmers via code stylometry, in: 24th {USENIX} Security Symposium ({USENIX} Security 15), 2015, pp. 255–270.
- [3] G. Frantzeskou, E. Stamatatos, S. Gritzalis, S. Katsikas, Source code author identification based on n-gram author profiles, in: IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer, 2006, pp. 508–515.

- [4] A. Fadel, H. Musleh, I. Tuffaha, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, P. Rosso, Overview of the PAN@FIRE 2020 task on Authorship Identification of SOURCE CODE (AI-SOCO), in: Proceedings of The 12th meeting of the Forum for Information Retrieval Evaluation (FIRE 2020), 2020.
- [5] E. Bogomolov, V. Kovalenko, A. Bacchelli, T. Bryksin, Authorship attribution of source code: A language-agnostic approach and applicability in software engineering, arXiv preprint arXiv:2001.11593 (2020).

## A. Online Resources

The sources for the implementation are available via

- Clang
- Tf-idf.
- Source code link.