SSNCSE_NLP@Authorship Identification of SOurce COde (AI-SOCO) 2020

Nitin Nikamanth Appiah Balaji, B. Bharathi

Department of CSE, Sri Siva Subramaniya Nadar College of Engineering,Tamil Nadu, India

Abstract

As the amount of data and software applications increases, it becomes important to identify the true authors for ownership and liability of the work. Issues such as plagiarism in academic activities, open-source contributions, and identification of the creators of malware applications can be done using automatic authorship identification models. In this work, the performance of Character Count vectorization and TFIDF models are studied on the AI-SOCO data-set. We achieved a significant improvement from the baseline with 85% accuracy on the test-set and 92% accuracy on the dev-set.

Keywords

Natural Language Processing, Plagiarism detection, Machine Learning

1. Introduction

The authorship identification can solve two major problems: identification of plagiarism by interpreting a program writing based on the previous works, and identification of the creators of malicious applications. Both are equally intimidating and the only solution is to device a foolproof model for identifying the author of a particular code snippet by understanding the users' style in writing programs. This model aims to safeguards the rights of one's work in academics, open-source contributions, projects, online coding contests, and all around the open internet.

Authorship detection on academic activity is important, for the identification of students cheating on their assignments. With online classes and online submission systems being promoted, it adds up to the need for the detection of plagiarism. In addition to this, recruitment processes for companies opting for online coding rounds escalate the need for identification of cheating. As other alternatives such as invigilation during academic exams and proctoring during recruitment processes are resource-intensive and require a large labor force, authorship identification becomes an efficient alternative.

Malware software is annoying enough to ruin a person's or even an entire organization's time. Even though laws against malware applications exists, it is hard to find the person involved in the program, to stop further malicious programs, and punish the person. Devising author identification models could instill fear in the first place, hence reducing the attempt for even indulging in coding malicious programs.

FIRE 2020: Forum for Information Retrieval Evaluation, December 16-20, 2020, Hyderabad, India
EMAIL: nitinnikamanth17099@cse.ssn.edu.in (N. N. A. Balaji); bharathib@ssn.edu.in (B. Bharathi)
ORCID: 0000-0002-6105-0998 (N. N. A. Balaji); 0000-0001-7279-5357 (B. Bharathi)

Table 1
Data Distribution

Data Set	Number of users	Number of programs per user	Total samples
Train-set	1000	50	50,000
Dev-set	1000	25	25,000
Test-set	1000	25	25,000
Total	1000	100	100,000

In this work feature extraction techniques and machine learning modeling techniques are analyzed. Char count vectorization with the Random Forest model is proposed for the authorship identification task. The AI-SOCO data-set containing C++ programs and corresponding user-id is used to analyze the performance of the models.

The remaining of the paper is organized in the following fashion: The data-set description and baseline analysis in Section 2, followed by the model architecture in Section 3, results and discussion in Section 4 and conclusions in the Section 5.

2. Data-set Description and Baseline Analysis

The data-set is a collection of C++ programs by 1000 users from the *codeforces* programming platform. The programs are coded in different versions of C++. It contains 100 (50 for training, 25 for development, and 25 for testing) different code snippets of each user. It contains a mapping of user-ids (uid) to the program-ids (pid) and the set of programs. These program files are verified and ready to compile, working programs that were submitted to the coding platform. As the data-set contains an equal number of samples for each testing individual, the data-set is fairly distributed, without any imbalance. The distribution of the data-set among the train, test, and dev-set is explained in Table 1.

The baseline model considered by the AI-SOCO challenge is a char Count Logistic regression model which gives an accuracy of 29.252%. In addition to this model, a TFIDF feature extractor with KNN based baseline with 10k features and k=25 is considered. This model shows an accuracy of 62.128% [1].

3. Architecture and Evaluation Scheme

The architecture included three parts - feature extraction, classification, performance evaluation. The C++ program file contains raw code, hence Count and TFIDF vectorizers are considered for feature extraction. These vectors and their corresponding uids now become a numerical classification problem, which is trained with Naive Bayes (NB) and Random Forest (RF) classifiers. As the data-set is well balanced the accuracy score is considered for comparing and evaluating the performance of the model. The scikit-learn [2] text feature extractor is used for converting the text into numerical vectors and scikit-learn's [2] RF and NB implementations are used for classification.

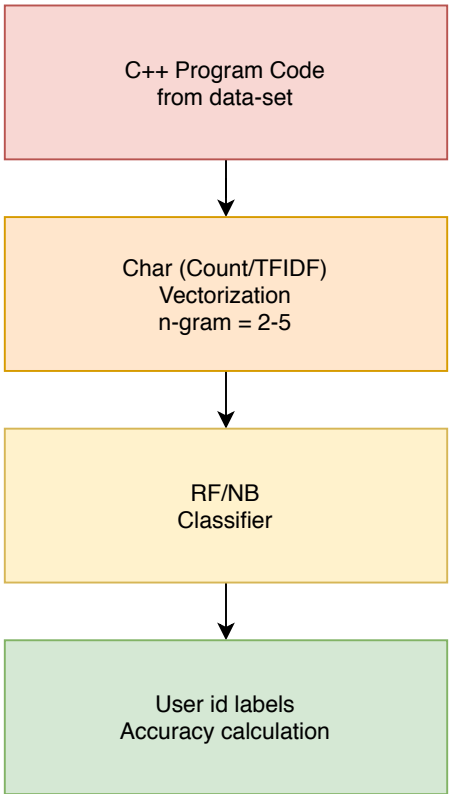

Figure 1: Steps involved in feature extraction and classification.

3.1. Count Vectorization

The count vectorization provides a primitive but powerful and simple way to transform the program documents to numerical vectors, as could be seen from previous work [3]. The count vectorization generates vectors of length equal to the length of the vocabulary the vectorizer is trained on and each value presents the number of instances the particular character or word appears in a document. This is helpful in the case of author prediction as some authors would mostly use a particular set of variable names for frequent mundane tasks. For instance for looping some show preference to *for* loops, whereas some prefer *while* loops, so is the bias with *if-else* and *switch-case* statements. Hence the set of characters or sequences of character difference is an important factor for studying a person's individual style of programming.

The count vectorizer is trained on the AI-SOCO corpus and this transformation function is used to fit the classification model on the train-set. Character Count vectorization with different n-gram ranges was analysed and the range of 2-5 was the best performing. The random forest proved to be the best model for fitting the large sparse matrix generated by the count vector transformation.

Table 2
Dev-set performance

Features	n-gram range	Classifier	Accuracy
Char Count vec	2-5	RF	**0.9211**
Char Count vec	2-5	NB	0.8108
Char TFIDF vec	2-5	RF	0.9160
Char TFIDF vec	2-5	NB	0.8128
Baseline TFIDF	-	KNN	0.6212
Baseline Count	-	Logistic	0.2925

3.2. TF-IDF vectorization

Term Frequency Frequency Inverse Document Frequency (TF-IDF) method is an extension of the Count vectorization technique. This method has shown significant results with feature extraction from program codes [4, 5]. The particular difference between the two techniques is that the TF-IDF method has an additional inverse frequency term to give importance to the rare words/characters in the documents. It gives a weight for each character based on its frequency of occurrence in all the provided documents. This reduces the significant impact of too banal words on the vector output. This could be of great importance as the common syntax of a C++ program includes the tokens and keywords like *main, struct, class, int, float, etc.,* These are used by all the users and could be of lesser importance and hence is removed for concentrating only on the important lexicons.

Similar to the count vectorizer, the TF-IDF vectorizer also shows good performance with an n-gram range of 2-5 with a random forest classifier.

4. Results and Observations

In this section the vectorization techniques are compared based on the performance on the dev-set and test-set accuracy scores. Both the techniques generated a significant improvement when compared to the baseline results. The random forest classifier is observed to perform better than the Naive Bayes classifier. With respect to the Char Logistic regression model there is a huge increase in accuracy from 29% to 92% on the dev-set. Similarly with respect to the TFIDF-KNN baseline model there is a increase in accuracy from 62% to 92% with around 48% improvement on the dev-set. The detailed scores on the dev-set and test-set are shown in Table 2 and Table 3 respectively.

5. Conclusion

Authorship identification from program code is becoming important in recent times due to the rapid increase in the number of free-to-use applications, online academic and research contributions. This could help to regulate plagiarism and detect the creators of malware softwares. In this work, Char count based feature extraction technique along with a Random

Table 3
Test-set performance

Features	n-gram range	Classifier	Accuracy
Char Count vec	2-5	RF	**0.8573**
Char TFIDF vec	2-5	RF	0.8500

Forest classifier is proposed for the AI-SOCO data-set. Our system showed significant increase in performance with respect to the baseline model with an accuracy of 85% on the test-data and 92% on the dev-set.

References

[1] 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), CEUR Workshop Proceedings, CEUR-WS.org, 2020.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[3] A. Ramırez-de-la Cruz, G. Ramırez-de-la Rosa, C. Sánchez-Sánchez, W. Luna-Ramırez, H. Jiménez-Salazar, C. Rodrıguez-Lucatero, Uam@ soco 2014: Detection of source code reuse by means of combining different types of representations, FIRE [4] (2014).

[4] S. Phani, S. Lahiri, A. Biswas, Personality recognition in source code working note: Team besumich., in: FIRE (Working Notes), 2016, pp. 16–20.

[5] M. Giménez, R. Paredes, Prhlt at pr-soco: A regression model for predicting personality traits from source code., in: FIRE (Working Notes), 2016, pp. 38–42.