Boosting a kNN classifier by improving feature extraction for authorship identification of source code

Yves Bestgen[a]

[a]Université catholique de Louvain, 10 place Cardinal Mercier, Louvain-la-Neuve, 1348, Belgium

Abstract

This paper presents the system developed by the LAST to identify the author of a source code. It combines the classical 1-nearest neighbor algorithm with a feature extraction step whose main characteristics are the use of indentation-aware tokenization to gather n-grams and skip-grams, which are weighted by Relevance Frequency. Its performance is over 92% correct identification of the author among 1,000 potential programmers.

Keywords

1-nearest neighbor, indentation-aware tokenization, skipgrams, relevance frequency supervised weighting

1. Introduction

This paper presents the system developed by the Laboratoire d'analyse statistique des textes (LAST) for the PAN@FIRE 2020 Task on Authorship Identification of SOurce COde (AI-SOCO). This task aim is to identify the programmer who wrote a source code. As stressed by the challenge organizers [1], being able to perform this task effectively should make it possible to combat cheating in academic and scientific communities. It could also help to fight against copyright infringement and intervenes in code integrity investigations [2].

Authorship identification is a question that has held attention for many years, especially in the literary [3, 4, 5] and history fields [6]. More recently, due to the development of computers and the Internet, works have focused on plagiarism and the re-use of sources without citing them adequately. Identifying the author of a literary text or a source code are two relatively different tasks, not only because of the differences between these genres of documents [2], but also because the potential authors of a literary work are usually well known and in very small numbers while the number of potential programmers is very large. This is for example the case in the present shared task since no less than 1,000 programmers must be discriminated. This task, therefore, appears a priori to be particularly complex.

As in other areas of machine learning, the state-of-the-arts approach is based on Deep Learning [7, 8, 9]. However, given the very large number of potential authors in this challenge, the classical k-nearest neighbors (kNN) algorithm could prove to be very effective. In line with my previous research ([3] for example), the main objective of the study is thus to try developing the most efficient *no deep learning* system without using any additional resources such as pre-trained language models or knowledge bases. To do this, special attention was put on tokenization, by trying to take into account the source code format style [10], on the feature set used, which includes skipgram [11], and

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

yves.bestgen@uclouvain.be (Y. Bestgen)

https://perso.uclouvain.be/yves.bestgen (Y. Bestgen)

0000-0001-7407-7797 (Y. Bestgen)

on the weighting of these features, by adapting *relevance frequency*, the supervised weighting scheme proposed by [12].

The rest of this paper describes the materials collected by the challenge organizers, the system developed, and the level of performance it has achieved. It also presents an analysis of the usefulness of each of its components.

2. Data and Challenge

The organizers of the AI-SOCO task collected from the Codeforces site (http://codeforces.com), a web platform on which programming contests are organized, 100,000 source codes in C++, 100 codes for each of the 1,000 selected participants. These source codes have all been considered to be correct by Codeforces. [1] provide useful statistics about this dataset.

This dataset has been divided into three sets: the training set composed of 50 source codes for each of the 1,000 authors, the development set consisting of 25 source codes for each author, and the test set also made up of 25 source codes for each author.

Author IDs were provided for the training and development sets, but not for the test set. Strangely enough, the rules of the challenge forbade adding the development set to the training set to predict the test set while the answers were provided. During the testing phase, each team could submit three solutions that were evaluated using the accuracy metric, a logical choice given that the number of instances in all categories is the same.

3. Developed System

The specificity of the developed system is clearly not in the supervised classification procedure used since it is the classical kNN algorithm, but in the features and their weighting. The choices made to try to arrive at the best possible classifier are described below. Each option was evaluated based on a stratified sample of 5,000 instances of the development set (instead of the full set to speed up the evaluation). From time to time, certain choices were also evaluated on the full development set and on the training set using a Leave-one-out cross-validation technique (LOOCV) [13, 14]. Document preprocessing and feature extraction were performed using the Statistical Analysis System (SAS). The kNN algorithm was performed using a custom C program [15].

3.1. Indentation-Aware Tokenization

As underlined in the first works in source code authorship identification [16] (see [17] for a synthesis), programmers can be discriminated by source code components such as preferred variable names and ways of doing typical operations, but also by the code layout such as indentations and space between and within instructions. For this reason, the tokenization step kept track of indentations. Concretely, the contiguous spaces and tabs present at the beginning of a line are considered as tokens in the same way as any other sequence of characters separated by space or tab. For example, the following two lines (> and < marking the start and end of a line):

```
>        for(auto &p : elem.second) {<
>            i = p.first;<
```

produce the following tokens (> and < marking the start and end of a token):

```
>        <
>for(auto<
>&p<
>:<
>elem.second)<
>{<
>             <
>i<
>=<
>p.first;<
```

In this tokenization, the space and the tab are distinguished from each other.

3.2. Features

The extracted features are composed of token n-grams with n ranging from 1 to 4. To reduce the size of the data, n-grams longer than 40 characters, an arbitrary threshold, were discarded. As indentations are considered as "normal" tokens, they are taken into account in the n-grams.

Skipgrams [11], n-grams of which one or several tokens located in the middle are replaced by a placeholder, were also extracted, one from each 3-gram (*a_b_c* producing *a_?_c*) and three from each 4-gram (*a_b_c_d* producing *a_?_b_c*, *a_b_?_d* and *a_?_?_d*). Minimum frequency thresholds were applied to these different types of features: 30 for 1-grams, 50 for 2-grams and 40 for 3-grams, 4-grams and skipgrams.

3.3. Weighting

Instead of the classic tf.idf frequently used in the field (for example [7]), the system proposed here uses *relevance frequency* (*rf*), the supervised weighting factor of [12], which favors features that are positively associated with the target category. Its formula is

$$rf = log_2(2 + \frac{a}{max(1, c)}),$$

where a is the number of instances in the target category in which this feature occurs and c is the number of instances in all other categories in which this feature occurs. As this is a supervised weighting scheme, it is computed on the training set only. This *rf* weighting is combined with a binary coding (presence or absence) of the feature.

Since the category of an instance is ignored for the test set, [12] proposed to use for weighting the unknown instance the weights of the category with which it is compared. If this approach is technically feasible, it poses practical problems since it is necessary to reweight the features of an unknown instance as many times as there are potential categories. The approach proposed by [18] for other supervised weighting procedures was used instead. It consists of using the maximum weight that the feature in question has received on all the categories considered in the learning set. The *rf* weights used for the test set are therefore calculated by the same formula and on the same data as the weights used for the training set.

3.4. Adding Centroids

While it was not obvious that this factor could be of much use, the centroid of each category was added to its 50 instances in the training set. It is obtained by calculating the mean weight of all the

features present in at least one of these 50 instances.

3.5. L2 Normalization

The analyzes showed that the use of L2 normalization markedly improves the system performance.

3.6. Parameters for the kNN Algorithm

The analyzes carried out on the development set led to fix the kNN parameter k at 1 and to use the classical Euclidean distance.

4. Analyzes and Results

4.1. Performance on the Test Set

As allowed by the rules of the challenge, three submissions were made on the test set. The first was based on the system described above and obtained an accuracy of 0.9217 while its accuracy on the development set was 0.9269. The other two systems submitted tried to account for the information provided by the challenge organizers that the test material contained exactly 25 instances of each of the 1,000 categories. More precisely, the classification was carried out by employing the best matching first approach (see [19] for the use of this approach in a case of a pair-matching) and by stopping assigning instances to a category when a sufficiently large number of instances had been assigned to it. As this kind of information is obviously not available in real life and as these two submissions scored only very slightly higher than to first one (0.9219 instead of 0.9217), they are not discussed in more detail.

The system ranked fourth in the challenge out of 16 teams with 0.0294 less than the first (0.9511), 0.0119 less than the 3rd (0.9336) and 0.006 more than the 5th (0.9157). It is also interesting to compare its performance with the TF-IDF kNN Baseline proposed by the organizers, which is based on the standard sklearn tokenizer, a TF-IDF weighting applied to the 10,000 most frequent features, and a kNN classifier with $k = 25$. It scored 0.6278 on the test material, 0.2939 lower than the system proposed here.

4.2. Factors Affecting the System Performance

If the supervised learning procedure is very classic, the feature extraction is somewhat different from what is usually done in the field. It is therefore useful to discuss the benefits provided by its different components. To do this, Table 1 compares the full system described in section 3 (the system used for the first submission) to a series of other systems obtained by ablation, modifying one component at a time. The full system is therefore based on the indentation-aware tokenization, the n-grams (n ranging from 1 to 4) and the skipgrams, the rf weighting, adding the centroids of the categories and using $k = 1$ while for building the other systems only one of these components is modified like the tokenization procedure, the weighting or the value of k. The accuracy scores reported in Table 1 were obtained on a stratified sample of 5,000 instances taken from the development set.

Table 1 indicates that three factors have an important impact on performance:

- Tokenization which, when done in the classical way, by simply separating tokens according to the presence of space or tab, results in a performance decrease of 0.0464.

Table 1
Ablation Analysis of the Usefulness of the System Components

Systems	Accuracy
Full system	0.9244
Standard tokenization	0.8780
Without skipgrams	0.9218
1&2&3-grams	0.9194
1&2-grams	0.9080
1-grams	0.8890
Tf.idf weighting	0.9192
Without centroids	0.9238
k = 3	0.9222
k = 5	0.9160
k = 9	0.9030
k = 25	0.8752

- The extracted features since the system based only on 1-grams obtains an accuracy lower by 0.0354 compared to the full system. Adding each additional n-gram length improves performance, but the gain provided by the 4-grams is small and less than that provided by the skipgrams. It should be remembered, however, that three of the four types of skipgrams are based on the 4-grams.
- The value of k has also a very important effect. The smaller the k, the better the performance, a k of 25, as in the baseline proposed by the organizers, producing a loss of 0.0492.

The other two factors have a much smaller impact. This is particularly the case with the addition of centroids (+0.0006). The *rf* weighting brings a gain of 0.005, which is a bit disappointing.

4.3. Qualitative Analysis

At first glance, it seemed difficult to imagine that it would be possible to achieve such a high level of performance (let alone the level reached by the challenge best system) in a task that requires identifying 1,000 different programmers based on 50 instances for each of them. So, I tried to understand why this task was "so" simple. The main reason is that the vast majority of programmers in the datasets do not try to hide their identity. Many of them systematically insert at the beginning or at the end of all or almost all of their source codes metadata stored in block comments that distinguish them from all the others. Many programmers also systematically insert preambles (idiosyncratic declarations and insertion of standard declaration files) at the start of the code, which they systematically copy from one program to another and therefore whose content and order are always identical.

There is, however, one special case that deserves mention here. Analysis of the errors made by the system shows that it was unable to distinguish two programmers from each other, User IDs 91 and 147. The system makes more than 50% of errors on them, the errors made on one consisting of assigning the source code to the other programmer and vice versa. The comparison of the source codes of these two programmers shows that they both use very frequently the same metadata and preambles around the C++ instructions needed to solve the problem as shown in Figure 1. Without further information, it is difficult to understand the origin of this similarity.

uid = 91, pid = 74855

```
///   _   _       _    _  _
///  |_  |_ |_| |_ |\   _|  |
///   _| |_  |  |_ |/   _|  |
///
#include <bits/stdc++.h>

#define ss second
#define ff first
#define pb push_back
#define faster ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
#define error(args...) {string _s=#args;replace(_s.begin(),_s.end(),',',' ');s

using namespace std;

void err(istream_iterator<string> it){}
template <typename S37,typename... Args>
void err(istream_iterator<string> it,S37 a,Args... args){cerr<<*it<<" = "<<a<<

typedef long long ll;
typedef long double ld;

const long long N=300100,mod=1e9+7,mod2=1e9+9,mod3=998244353,sq=450,base=727,lg

ll n,m,x,y,w,z,t,k,d,ans,a[N],b[N];

string s,f[N];

int main(){
 faster;
    cin>>n>>y>>x>>s;
    reverse(s.begin(),s.end());
    for(int i=0;i<y;i++){
        if(i==x)s[i]='1'-s[i]+'0';
        if(s[i]=='1')ans++;
    }
    cout<<ans;
    return 0;
}
/*
     Welcome To My Maze!
  ______________________
 |   |_  _  |___    |___
 | |  __| | |__ | | |   | |
 | |__  | |  ___| | | |_| |
 | _  | | |  _____| | |   |
 |_ |___  | |  ___  | | | |
 |_____ | | | |   |  _| | |
 |__    | | | | |_| |  _| |
  ___|____|___|_______|___|

*/
```

uid = 147, pid = 35348

```
///   _   _       _    _  _
///  |_  |_ |_| |_ |\   _|  |
///   _| |_  |  |_ |/   _|  |
///
#include <bits/stdc++.h>

#define ss second
#define ff first
#define pb push_back
#define faster ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
#define error(args...) {string _s=#args;replace(_s.begin(),_s.end(),',',' ');st

using namespace std;

void err(istream_iterator<string> it){}
template <typename S37,typename... Args>
void err(istream_iterator<string> it,S37 a,Args... args){cerr<<*it<<" = "<<a<<e

typedef long long ll;
typedef long double ld;

const long long N=300100,mod=1e9+7,mod2=1e9+9,mod3=998244353,sq=450,base=727,lg

ll n,m,x,y,w,z,t,k,d,ans,a[N],b[N];

int main(){
 faster;
 cin>>n>>m;
    for(int i=0;i<n;i++){
        cin>>a[i];
        x=max(x,a[i]);
...
        ans+=a[n-2];
        ans-=a[n-1];
    }
    cout<<ans;
    return 0;
}
/*
     Welcome To My Maze!
  ______________________
 |   |_  _  |___    |___
 | |  __| | |__ | | |   | |
 | |__  | |  ___| | | |_| |
 | _  | | |  _____| | |   |
 |_ |___  | |  ___  | | | |
 |_____ | | | |   |  _| | |
 |__    | | | | |_| |  _| |
  ___|____|___|_______|___|

*/
```

Figure 1: Source codes from two different programmers.

5. Discussion and Conclusion

The system developed by the LAST for the PAN@FIRE 2020 Task on Authorship Identification of SOurce COde (AI-SOCO) was aimed at obtaining the best possible performance by combining a supervised learning algorithm among the most classical with a feature extraction step as efficient as possible. The classifier used is the 1-nearest neighbor algorithm. The features are extracted by a fully automatic procedure exclusively from the training set. The performance achieved is over 92% correct identification of the author of a source code among 1,000 potential programmers, ranking the system 4th out of 16 teams. Analyzes carried out on a sample of the development set indicate that the indentation-aware tokenization, the extraction of n-grams from 1 to 4 tokens and skipgrams, and setting the parameter k to 1 are at the source of the efficiency of the system, as well as, but to a lesser extent, the use of *relevance frequency*, the supervised weighting scheme proposed by [12]. The performance obtained on the test set seems rather high for a system that is based on an approach as simple as kNN and which only uses training data without the addition of external knowledge or precomputed embeddings.

An important weakness of the kNN algorithm is that it is biased in favor of the most frequent categories. In the present task, this is not a problem since the categories are balanced, but this weakness should be kept in mind if the system is used with other datasets in which this condition is not met.

Another limitation of the system is that it is far from obvious that it could prove useful for the purpose for which the AI-SOCO task was designed: to aid in the detection of cheating in the academic community (https://sites.google.com/view/ai-soco-2020/task-description). The proposed system benefits too much from the fact that the vast majority of programmers do not try to hide their identity, systematically inserting at the beginning or at the end of all or almost all of their source codes, metadata which distinguishes them from all the others. When two different programmers use the same metadata in their source code the system performance is very low. It would be interesting to determine what level of performance the proposed system could achieve when using only the parts of the source codes that contain these metadata or, even more interesting, to determine the efficiency of this system, but also of the other systems participating in this task [1], on source codes without these metadata.

Regarding possible developments, it could be very fruitful to try to make tokenization more efficient since the analyzes have shown that even the partial use of code formatting is very beneficial. Taking inspiration from [9] could be particularly relevant.

Acknowledgments

The author wishes to thank the organizers of the AI-SOCO task for putting together this valuable event and the two anonymous reviewers for their very helpful comments. He is a Research Associate of the Fonds de la Recherche Scientifique - FNRS (Fédération Wallonie Bruxelles de Belgique). Computational resources have been provided by the supercomputing facilities of the Université Catholique de Louvain (CISM/UCL) and the Consortium des Equipements de Calcul Intensif en Fédération Wallonie Bruxelles (CECI).

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] C. Zhang, S. Wang, J. Wu, Z. Niu, Authorship identification of source codes, in: L. Chen, C. S. Jensen, C. Shahabi, X. Yang, X. Lian (Eds.), Web and Big Data - First International Joint Conference, APWeb-WAIM 2017, Beijing, China, July 7-9, 2017, Proceedings, Part I, volume 10366 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 282–296. doi:10.1007/978-3-319-63579-8_22.

[3] Y. Bestgen, Recherche en paternité littéraire : deux romans en quête d'auteur, in: V. Martin-Schmets (Ed.), Henri Vandeputte : Oeuvres complètes, volume 3, Tropismes, Bruxelles, 1992, pp. 379–404.

[4] W. M. A. Smith, The revenger's tragedy: The derivation and interpretation of statistical results for resolving disputed authorship, Computer and the Humanities 21 (1987) 21–55.

[5] H. H. Somers, Analyse statistique du style, Nauwelaerts, Louvain, 1962.

[6] F. Mosteller, D. L. Wallace, Inference in an authorship problem, Journal of the American Statistical Association 58 (1963) 275–309.

[7] M. Abuhamad, T. Abuhmed, A. Mohaisen, D. Nyang, Large-scale and language-oblivious code authorship identification, in: D. Lie, M. Mannan, M. Backes, X. Wang (Eds.), Proceedings of the

2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, ACM, 2018, pp. 101–114. doi:10.1145/3243734.3243738.

[8] E. Bogomolov, V. Kovalenko, A. Bacchelli, T. Bryksin, Authorship attribution of source code: A language-agnostic approach and applicability in software engineering, CoRR abs/2001.11593 (2020).

[9] S. Gorshkov, M. Nered, E. Ilyushin, D. Namiot, V. Sukhomlin, Source code authorship identification using tokenization and boosting algorithms, in: V. Sukhomlin, E. Zubareva (Eds.), Modern Information Technology and IT Education, Springer International Publishing, 2020, pp. 295–308.

[10] I. Krsul, E. H. Spafford, Authorship analysis: identifying the author of a program, Comput. Secur. 16 (1997) 233–257. doi:10.1016/S0167-4048(97)00005-9.

[11] Y. Wilks, Reveal: the notion of anomalous texts in a very large corpus, Tuscan Word Centre International Workshop, Certosa di Pontignano, Italy, 2005.

[12] M. Lan, C. L. Tan, J. Su, Y. Lu, Supervised and traditional term weighting methods for automatic text categorization, IEEE Trans. Pattern Anal. Mach. Intell. 31 (2009) 721–735. doi:10.1109/TPAMI.2008.110.

[13] P. A. Lachenbruch, R. M. Mickey, Estimation of error rates in discriminant analysis, Technometrics 10 (1968) 1–11.

[14] Y. Bestgen, S. Granger, J. Thewissen, Error patterns and automatic l1 identification, in: S. Jarvis, S. Crossley (Eds.), Approaching Transfer through Text Classification: Explorations in the Detection-based Approach, volume 3, Multilingual Matters, Bristol, UK, 2012, pp. 127–153.

[15] Y. Bestgen, Détermination de la valence affective de termes dans de grands corpus de textes, in: Actes du Colloque International sur la Fouille de Texte CIFT'02, INRIA, 2002, pp. 81–94.

[16] I. Krsul, Authorship analysis: identifying the author of a program, Department of Computer Science Technical Reports TR-94-030, Purdue University, 1994.

[17] S. D. Burrows, Source code authorship attribution, Ph.D. thesis, RMIT University, Melbourne, 2010.

[18] I. Batal, M. Hauskrecht, Boosting KNN text classification accuracy by using supervised term weighting schemes, in: D. W. Cheung, I. Song, W. W. Chu, X. Hu, J. J. Lin (Eds.), Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009, ACM, 2009, pp. 2041–2044. doi:10.1145/1645953.1646296.

[19] Y. Bestgen, LSVMA : au plus deux composants pour apparier des résumé à des articles, in: Actes de DEFT2011 : septième Défi Fouille de Textes, ATALA, 2011, pp. 105–114.