Automated Tools for Source Code Plagiarism Detection

Kristian Wahlroos

Seminar report
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, March 20, 2017

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Kristian Wahlroos

Työn nimi — Arbetets titel — Title

Automated Tools for Source Code Plagiarism Detection

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminar report	March 20, 2017	6

Tiivistelmä — Referat — Abstract

Plagiarism is a serious problem which can be hard to detect. Especially in massive online courses (MOOC), where students are required to complete programming tasks, there is no other guarantee than trust, that the student has really completed his/her assignment completely by themselves. Same kind of situation is possible in academic world, where universities held courses that relies on students completing various programming tasks. This setting can lead to cases of plagiarism where the solution for a given exercise is actually copied from a study friend or from the Internet. Also obfuscation is often used by plagiarists to hide their cases of plagiarism. This makes detecting plagiarism even harder and with the sheer number of participant, the plagiarism detection is almost impossible to do as a manual labor.

This paper describes the current solutions for automated plagiarism detection from the source code by doing a literature review. The focus will be on the machine learning aspect and the scenario that plagiarism is reflected to, is a scenario where student is completing programming assignments for a grade. This kind of scenario is very common one among universities, and exists also at University of Helsinki's course '*Introduction to programming*'.

Avainsanat — Nyckelord — Keywords
Machine learning, Source code, Plagiarism, Author identification

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Massive online courses or *MOOCs* have gained a lot of popularity in recent years. Their course formats are easy to follow and don't require any live attendance from the students to be able to complete the courses. MOOCs also has a quite flexible schedules to complete them, which makes them very interesting in the eyes of a student. Some MOOCs like edX[1] and Coursera can even offer real credits for the students, if students universities have made a deal with them. From available MOOCs, programming has gained a lot of interest and in the end of 2016, edX even listed three programming courses as the years most popular courses[2]. The course formats of programming MOOCs work more or less the same way; videos for lectures and returnable programming assignments as tasks. These programming assignments are usually automatically tested and credited for the student, and this makes it very easy for the student to complete programming courses from their home computers whenever they want.

Many MOOCs work by trusting that the student has completed exercises by themselves and this is not really guaranteed in any way. In other words there usually doesn't exists any automatic system for possible plagiarism detection inside these courses. This leaves MOOCs in a rather difficult position, because they offer credits based on one-way trust, and without a final exam with mandatory attendance, one-way trust becomes a quite shallow. For example, source code plagiarism is relatively easy and without automatic tools it's almost impossible to get caught if there are hundreds or even thousands of students. Detecting source code plagiarism is thus basically impossible for the course keepers without any automated tools.

In this paper tools and methodologies for automated source code plagiarism detection is reviewed. The scope will be focusing on various machine learning techniques, while the reflected scenario is kept to be at MOOC-style courses or academic world where plagiarism could happen within the normal programming course with assignments. Also the term *machine learning technique* is considered as a term, that covers following subcategories: unsupervised learning, supervised learning, data mining and various probabilistic models. Thus it's not limited to only to analyze e.g. labeled data, meaning already known cases of plagiarism in one scenario.

The structure of the rest of this paper will be following: starting with the background motivation follows the methodology part where techniques for the literature review is described; then comes the results where different techniques and features from various papers is described; analyzing part sums up those findings to find common cases or really differing techniques; finally comes discussion part that reflects the findings on how could these be

[1] See more at `https://www.edx.org/credit`

[2] Based on `http://blog.edx.org/10-most-popular-courses-edx-courses-in-2016`

used at programming related MOOCs.

1.1 Background

As there are countless of opportunities in MOOCs to plagiarise source code, there are as equal amount of ways to do it in academic courses. Usually plagiarism in courses is not necessary even intentional but just by-product of course friends doing the same tasks together. For example in *Introduction to programming*[3] course organized by University of Helsinki, the course lasts 7 weeks and every week student has to complete programming assignments with Java-language. Every student has same assignments in the same order and they are encouraged to help each others during sessions. This naturally can create problematic cases of group work, which as recurring events are also real plagiarism in a form of source code sharing. One solution in here would be automatically detect those cases and then try to resolve them manually.

Other types of source code plagiarism are direct plagiarism and obfuscated plagiarism. Direct plagiarism in source codes is basically copy-pasting the code and it's the most simplest form of plagiarism. Obfuscation means that the plagiarist is trying to hide his/her plagiarism by making the code look more like it was made by him/her. In source code, obfuscation could relate to renaming variables and functions or making slight modifications into the plagiarised code e.g. changing the order or making slight modification into the logic of the code.

2 Methodology

The references for this paper were gathered by performing systematic literature review utilizing *Google Scholar*. This was done in a two step manner where first a collection of papers were gathered by searching with specific keywords, then these papers were filtered if their abstract, keywords, title or introduction contains specific terms.

In the first step, used search terms which required a direct match were **machine learning**, **plagiarism** and **code**. Other terms used were **authorship** and **identification** but these didn't require a direct match from the results. Papers were filtered also by year, only considering papers from 2006 onward. This was done because most modern MOOCs like Coursera and Udacity are relatively new, and automated programming assignments haven't existed before. Papers from 2006 onward also have a high probability to consider modern programming languages e.g. Java or Python, which both are heavily used to teach programming in universities[4].

[3]This years course page at `https://2017-ohjelmointi.github.io/`

[4]Based on `http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext`

Second step contained a more specific filtering, where only papers that included words **plagiarism**, **programming** or **code** were selected. This was done to get more specific results which focuses more on the scope of this paper. That is, detecting source code plagiarism with machine learning.

Gathered papers are analyzed together to form the final results of this paper. This is done by looking at the methodologies and results of gathered studies by their accuracies, features extractions and usage of machine learning techniques. This is believed to help to give answers on how plagiarism could be detected from the source code, targeting especially academic courses and MOOCs.

3 Results

Machine learning in a source code plagiarism detection works like any other machine learning classification technique in another scenario. It is used to find recurring patterns from the data that could be used to separate it into individual parts. In source code plagiarism, this means finding features that could be used to identify the author and classify unseen data to most likely author. Unseen data in this sense means the source code from an unknown author, and the goal is to predict that which one of the possible authors wrote the unseen source code or is the unseen source code too similar to some other code.

Previous scenario is issued in [1], where Bandara and Wijayarathna are using attribute counting technique to develop a machine learning model to detect source code plagiarism. Attribute counting technique means, that the original source code is transformed into feature vector that contains only stylish metrics i.e. no data from the underlying structure. They report final accuracy as 86.65%. Same idea of using style metrics from source code was researched earlier by Lange and Mancoridis [6]. They use 18 various features and measure distribution differences of the metrics. Lange and Mancoridis report the final accuracy of their model as 75%. Also Kothari *et al.* take the coding style and character sequences as their features for the model [5]. They have six different style and text distribution metrics which can represent the distribution of n-character patterns[5]. Their reported accuracy when using students source codes is for the pure style metrics 36% and for the n-character 69%. Elenbogen and Seliya build a data mining model based on six different style metrics in [3]. They call these as a *programming profiles* and these style metrics are described as elements that doesn't affect the functionality. Their accuracy using 83 programs and 12 students was 74.7%.

Other studies use more structural analysis of the source code to form the features. In [4], Jadalla and Elnagar describe the *PDE4Java* detection engine to detect plagiarized Java-code. Authors use tokenization to build

[5]E.g. using 2 character patterns slice the source code into two character slices.

the needed format from the source code, that can be utilized for the N-Gram representation. This N-Gram representation is basically same as splitting the source code into slices of chosen length. Their data mining method reported suspicious source codes from eight various sized data sets about 6 out of 8 times in the same way as domain expert reported. The authors however didn't include the specific accuracy of the data mining model. Comparing to the study of Jadalla and Elnagar, even more structural-based analysis is done by Caliskan-Islam *et al.* They use various code stylometries derived from abstract syntax trees to build the model that can de-anonymize programmers [2]. Their model accuracy is 94% when there were 1 600 possible authors and 98% when there were 250 authors. Rosenblum *et al.* researches that does programming style preserve after compilation [7]. They use style metrics for authorship identification, but this time style metrics from the compiled code as original source code might not always be available, making the style metrics more structural based. Their validation used 20 authors and the model reaches 77% accuracy. Son *et al.* propose a parse tree kernel to form the model based on structural features in [8] using 555 source codes. Their reported accuracy was 93% same as using a human validator for the same task.

The collected papers seems to divide into two different categories: pure stylistic features [1, 6, 5, 3] and structural based features [4, 2, 7, 8]. As most stylistic metrics are based on easily collectable metrics from the original source code, structural approaches usually performs pre-processing to get the underlying features. Pre-processing is in many cases done by forming abstract syntax tree -representation which is not affected e.g. if names of the variables are changed.

3.1 Data

The data sets used and their sizes in previous studies vary a little bit; source codes from college students, synthetic data sets, codes collected from open source projects and codes from open competitions e.g. *Google Codejam*. Most variance comes from the number of possible authors and the numbers of used source codes, but it doesn't seem to matter data-wise is the method structural or style based. The findings related to data set sizes are summed into following table, where *size* refers to the total amount of files used.

Attr./Paper	[1]	[5]	[3]	[6]	[8]	[2]	[4]	[7]
Size	741	200	83	4068	555	N/A	326	203
Authors	10	8	12	20	N/A	1600	N/A	32

Table 1: Reported data sets used in papers.

In table 1, values are chosen only if they are explicitly told in the paper. If there have been multiple values or data sets, data set size have been selected

as the one with student-related data, which are data sets collected from the courses. In three studies explicit values are not told and that is why they are marked as *N/A*. For example in the study of Caliskan-Islam *et al.*, their data set was collected from Google Codejam which explains their large amount of programmers[2]. However they didn't explicitly tell how many source code files there were in total, but rather mentioned that there were around one to 17 code files per a contestant.

3.2 Methodologies

In this section, methodologies of collected studies are reviewed. Many style-related studies have similar methods, but their machine learning techniques or gathered feature sets differ. In structural-related, their methodologies can be very different but almost always utilizing the gained information from abstract syntax tree.

3.3 Feature extractions

3.4 Similarity detection

4 Analysis

Reflection

5 Discussion

Main points/main ideas. Limitations

6 Conclusion

Sum up findings.

References

[1] Bandara, Upul and Wijayarathna, Gamini: *A machine learning based tool for source code plagiarism detection.* International Journal of Machine Learning and Computing, 1(4):337, 2011.

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

[3] Elenbogen, Bruce S. and Seliya, Naeem: *Detecting outsourced student programming assignments.* J. Comput. Sci. Coll., 23(3):50–57, January 2008, ISSN 1937-4771. `http://dl.acm.org/citation.cfm?id=1295109.1295123`.

[4] Jadalla, Ameera and Elnagar, Ashraf: *Pde4java: Plagiarism detection engine for java source code: a clustering approach.* International Journal of Business Intelligence and Data Mining, 3(2):121–135, 2008.

[5] Kothari, Jay, Shevertalov, Maxim, Stehle, Edward, and Mancoridis, Spiros: *A probabilistic approach to source code authorship identification.* In *Information Technology, 2007. ITNG'07. Fourth International Conference on*, pages 243–248. IEEE, 2007.

[6] Lange, Robert Charles and Mancoridis, Spiros: *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*, pages 2082–2089. ACM, 2007.

[7] Rosenblum, Nathan, Zhu, Xiaojin, and Miller, Barton P: *Who wrote this code? identifying the authors of program binaries.* In *European Symposium on Research in Computer Security*, pages 172–189. Springer, 2011.

[8] Son, Jeong Woo, Noh, Tae Gil, Song, Hyun Je, and Park, Seong Bae: *An application for plagiarized source code detection based on a parse tree kernel.* Eng. Appl. Artif. Intell., 26(8):1911–1918, September 2013, ISSN 0952-1976. `http://dx.doi.org/10.1016/j.engappai.2013.06.007`.