Lowering the hesitation towards MOOC's by performing authorship attribution on python code comments

Romano Vacca

VU University, Amsterdam 1081 HV, The Netherlands,
r.d.vacca@student.vu.nl

Abstract. This paper covers the development of an algorithm that determines who the author of an python script is based on the comments within that script. Different from normal text classification, some new features had to be used and created. An existing project for normal text classification (NT) is used as a starting point, and adjusted and extended for source code text classification(SCC)[1]. In this project, NLTK and the sci-kit library have been used. The newly created algorithm scored an accuracy of 13,3% based on 15 python scripts, whereas NT classification on python script accuracy is 6,67%. Although this score is low, the SCC is an improvement of more then 100%. The algorithm used a voting mechanism to determine who the author is. Whenever two texts had the same score, the results of who the author was, was ruled as undecided and marked as not classified correctly.

Keywords: authorship attribution, mooc's, source code, classification, information retrieval

1 Introduction

Massive Online Open Courses (MOOC's) are the next big thing in education. McAuley et al. define MOOC's as: "a MOOC integrates the connectivity of social networking, the facilitation of an acknowledged expert in a field of study, and a collection of freely accessible online resources. Perhaps most importantly, however, a MOOC builds on the active engagement of several hundred to several thousand students who self-organize their participation according to learning goals, prior knowledge and skills, and common interests."[1].

Currently, there is a growing body of literature that recognizes the importance of these online courses, enabling people all around the world, regardless of where they are living, to receive the education they otherwise could never get. The only thing needed to make this happen, is a computer and an internet connection. Making its entrance in 2008, multiple universities have combined

[1] https://github.com/romanovacca/SCC-Authorship-Attribution

their efforts of producing high quality courses. For instance, edX[2] is a collaboration between Harvard[3] and MIT[4]. This initiative led to an online platform where one can receive certificates of completion or even credits counting towards one's degree. It was not until 2012, when an article published in the New York Times, stating that year to be "The year of the MOOC", that the world really got acquainted with this phenomenon[2].

However, some universities have doubts, since it is hard to determine to what extent a user actually did all the work for a a given online course. User identity authentication is one of the largest issues MOOC's have to deal with and there lies the foundation for this thesis[3]. The goal of this paper is to take some of the hesitation away from universities whether or not to participate in MOOC's. One of the solutions for this particular problem is using algorithms that analyze one's writing style to determine if that person actually is who he or she says he or she is. For plain text files or books, this can be achieved with high precision[4][5]. For programming languages it is less common. Most of these approaches are based on analyzing the coding style, whereas I will try to analyze the comments used within a Python script, to determine who actually wrote it. This program is not intended as a stand-alone solution but an addition to the previously mentioned approaches. I believe that by combining these different kind of attribution techniques, a higher trust and reliability towards MOOC's can be accomplished.

2 Related work

There are relatively few historical studies done on the subject of source code authorship attribution, since it is a fairly new topic. Burrows et al. performed source code authorship attribution on 1640 C language based documents with 100 different authors, achieving a 67% accuracy. The difference between that study and this one is that this paper focuses solely on the comments, whereas Burrows et al. choose to delete all the comments and measure similarity on the remaining code[6]. In a later attempt of Burrows et al., they managed to classify 76,78% correctly, but only in a 10-class authorship classification experiment[7]. Tennyson et al. managed to obtain a 98.2% accuracy with 7231 documents and 100 authors by using a Bayesian ensemble classifier[8]. While this sounds very promising, a remark they stated themselves was that this particular classifier could not be effective for student-submitted programs. One of the reasons is that in a teaching environment, teachers usually provide an assignment with sample code, leading to assignment with a large part of the same code. This then leads to skewed results.

[2] `http://www.edx.org`
[3] `http://www.harvard.edu`
[4] `http://web.mit.edu`

3 Methods

3.1 Software/techniques used

For this application, Python 2.7 was used. Python[5] was chosen because is it compatible with the open-source software Scikit-learn library, which makes it possible to identify authors by the use of machine learning algorithms[9]. Another tool used was NLTK[6]. NLTK stand for Natural Language Toolkit, which also is a Python library that helps analyzing and processing documents. Currently it is widely used in the computer science industry due to its usefulness[10].

3.2 Database creation

To create a database of scripts, fifteen random users from Kaggle[7] are selected. Kaggle is an online platform that hosts data science competitions. It is quite normal in the Kaggle community to publish the source code one uses during a competition. Other users may freely use this code and even try to improve it or include it in their own code. The accessibility of different source code by different authors is the reason why these scripts are used for this application. From the selected authors, only authors that have published at least four scripts will be chosen. There are two reason why I chose this small amount of scripts. The first reason is for simplicity of this project. The second is because when attending a course from an MOOC, a user can be asked to write short code prior to the real exercises of the course. This way, the specific way of writing of an author can be saved as a golden standard against other code. From the four scripts that are chosen, three scripts will be used as a training set, and the fourth will be the test set. Before the scripts can be analyzed, some preprocessing has to be done.

3.3 Preprocessing

First, the scripts have to stored as text files, so it can be processed in Python. The three training scripts from the user will be stored in one text file. The next step is to remove all the names or lines of code that state who the author of that script is. As the basis for this application, the project of NeilYager[8] is forked. Forking is using existing code, but adapting it to meet other requirements or other purposes[11]. The project of NeilYager consisted of a clustering algorithm that distinguished who the author was of a chapter of a book that was written by two authors. Although his project is used as a starting point, a number of things had to be adjusted. One of those changes had to do with the fact that the application of NeilYager used to distinguish authors from plain texts. From own experience, different from plain texts, is that people tend to use non-formal language in their comments. Because of this difference, the features used for

[5] `https://www.python.org`
[6] `https://www.nltk.org`
[7] `https://www.kaggle.org`
[8] `https://github.com/NeilYager/authorship-attribution`

classification in his projects also had to be altered, which now will be discussed further.

3.4 Features

Looking at plain text documents, such as chapters from a book, one can expect formal language with punctuation and proper grammar. People tend to use a user-specific amount of commas, semicolons, dots and certain words by which one can be distinguished. I did not expect this to be sufficient enough to determine the author from a non formal source code comment. This assumption was tested and is presented in the results section by comparing the accuracy of the application with the features used by NeilYager, and my own features developed specific for source code authorship attribution.

The version for normal text classification(NT) consisted of four features: Bag of words(BOW) feature, lexical feature, punctuation feature and syntactic feature. The bag of words feature considers the most common words in a text. The lexical feature looks at the structure of writing in a text. The punctuation feature measures how many times certain punctuation marks are used in a text. The syntactical feature keeps track of the distribution of the pos tags in a text.

Table 1. NT features

Bag of Words	the most common words in a text
Lexical	the structure of writing in a text
Punctuation	the use of punctuation marks such as "," and ";"
Syntactic	the distribution of the part of speech tags in a text

Table 2. SCC features

Tf-idf	Looks at how much the documents are the same
Lexical	the structure of writing in a text
Punctuation	the use of punctuation marks such as "," and ";"
Syntactic	the distribution of the part of speech tags in a text

In the Source Code Comments(SCC)[9] version, which was created for this project, the BOW feature is replaced by a Term frequency-inverse document frequency(Tf-ifd) feature. The motivation lies in the fact that short texts do not provide sufficient word occurrences, which put a limitation on the accuracy of classification methods[12]. However, Tf-ifd has proven to be very difficult to beat and has made its introduction into other retrieval areas than information retrieval[13]. The punctuation feature was adjusted by taking into account how

[9] `https://github.com/romanovacca/SCC-Authorship-Attribution`

many hashtags an author would use to denote the start of a comment. For the lexical feature, the number of different words used per comment was added.

4 Results

In this section, the scores of the NT version and the SCC version are compared. As input the algorithm takes in a training set from an author, then fifteen test sets are compared to it. For every script, a score is returned for the similarity between the authors use of bow, lexical, punctuation and syntactic features. The scores that are returned are either a zero or a one. Zero means that the tested script does not matches the authors style. A one means that the style from the tested script does resemble the authors style. This gives us four numbers for every test script, coinciding with the four features. In table 3 seen below an actual result is provided as example.

Table 3. Example of NTF classification for Author #1

Author	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
BOW	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0
Lexical	0	1	0	0	1	0	1	0	0	1	0	0	1	1	1
Punctuation	0	1	0	1	1	1	1	1	1	0	1	1	0	1	1
Syntactic	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1

Sum	2	4	1	2	3	1	2	2	2	3	2	2	2	2	3
Average	2,2														

In table 3, author #1 is used as a training set. It is then compared to all the test sets. For the results, a voting mechanism is used to determine who is most likely to be the author. Here, the training set of author #1 is most similar to the test set of author #2. This can be seen by the sum of 4 in the table.
Putting together all the intermediate classifications, the results of table 4 are obtained. Here, an classification is only considered as correctly predicted when

Table 4. The results for the NT and SCC algorithms

	NT	SCC
Percentage classified correctly	6,67%	13,3%
Average score per author	2,28	1,93

there is only one document that has the highest sum and that document matches the author of the training set. That means that if two documents have the highest sum, one of which actually is the same author as the training set, this still will be marked as not classified correctly.

5 Discussion and findings

Looking at the results, the SCC performed better than the NT. One of the reasons for this improvement was substituting the BOW feature for the Tf-idf feature. In table 5, the classification strength of that feature is showed. It managed to correctly predict 40 percent of the authors. This change is also the

Table 5. Classifications by Tf-idf feature

Author	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
Tf-idf	1	1	0	0	1	0	0	0	0	1	0	1	0	0	1

most important reason for the lower average of the SCC version compared to the NT version. The Tf-idf only returns a one for the most similar document, whereas the BOW feature returns a zero or a one per author. The lower average can also be very useful in the future. When the accuracy of this classifier is high, and a script returns a score above the average, this implicates that the writing style matches that of someone else. In other words, plagiarism is likely then.

The adjustments for the punctuation and lexical features were less successful. At the beginning, when looking at scripts, I expected that when adding the number of hastags used per author, that would be a very good feature. The results however, do not clearly confirm nor deny that. Adding it to the SCC version seems to be just as good as the other features that were already included in the NT version. Also, in python one can denote the start of a comment by using three apostrophes then some text and then three apostrophes again. This was left out of this research for simplicity and the small amount of time that was available to create this program. But by including this, more input from the can be used and be tested against, which could lead to a higher accuracy.

6 Conclusion

The purpose of this project was to develop an algorithm that can predict authors of python scripts, so universities would less be held back in joining MOOC's. To some extent, this project showed that it is possible to recognize an author of a python script by analyzing the comments. Even-though the accuracy of the SCC is not very high, it still is an improvement of more then a 100% compared to the NT version.

The Tf-idf seems to be a good feature, whereas the features used for normal classifications(lexical, syntactical, punctuation), are less fit. Developing other features is one of the main points that should be addressed to improve the accuracy of the SCC algorithm. When that is successful, this algorithm, along with the before mentioned stylistic algorithms, could together lead to a powerful tool in abolishing plagiarism of programming scripts and provide that last push for universities to follow this promising trend that is known as MOOC's.

References

1. McAuley, A., Stewart, B., Siemens, G., & Cormier, D. (2010). The MOOC model for digital practice. (4)
2. Pappano, L. (2012). The Year of the MOOC. The New York Times, 2(12), 2012.
3. Miguel, J., Caball, S., & Prieto, J. (2013, September). Providing Information Security to MOOC: Towards effective student authentication. In Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on (pp. 289-292). IEEE.
4. Elayidom, M. S., Jose, C., Puthussery, A., & Sasi, N. K. (2013). Text classification for authorship attribution analysis. arXiv preprint arXiv:1310.4909.
5. Ebrahimpour, M., Putni, T. J., Berryman, M. J., Allison, A., Ng, B. W. H., & Abbott, D. (2013). Automated authorship attribution using advanced signal classification techniques. PloS one, 8(2), e54998.
6. Burrows, S., & Tahaghoghi, S. M. (2007, December). Source code authorship attribution using n-grams. In Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University (pp. 32-39).
7. Burrows, S., Uitdenbogerd, A. L., & Turpin, A. (2009, April). Application of information retrieval techniques for source code authorship attribution. In Database Systems for Advanced Applications (pp. 699-713). Springer Berlin Heidelberg.
8. Tennyson, M. F., & Mitropoulos, F. J. (2014). A Bayesian Ensemble Classifier for Source Code Authorship Attribution. In Similarity Search and Applications (pp. 265-276). Springer International Publishing.
9. Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., & Mueller, A. (2015). Scikit-learn: Machine Learning Without Learning the Machinery. GetMobile: Mobile Computing and Communications, 19(1), 29-33.
10. Perkins, J. (2010). Python text processing with NLTK 2.0 cookbook. Packt Publishing Ltd. (pp. 18)
11. Ernst, N. A., Easterbrook, S., & MyLOPOULOS, J. (2010). Code forking in open-source software: a requirements perspective. arXiv preprint arXiv:1004.2889.
12. Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., & Demirbas, M. (2010, July). Short text classification in twitter to improve information filtering. In Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval (pp. 841-842). ACM.
13. Robertson, S. (2004). Understanding inverse document frequency: on theoretical arguments for IDF. Journal of documentation, 60(5), 503-520.