DCU@FIRE-2014: An Information Retrieval Approach for Source Code Plagiarism Detection

Debasis Ganguly
Centre for Global Intelligent Computing (CNGL)
School of Computing
Dublin City University
Dublin, Ireland
dganguly@computing.dcu.ie

Gareth J.F. Jones
Centre for Global Intelligent Computing (CNGL)
School of Computing
Dublin City University
Dublin, Ireland
gjones@computing.dcu.ie

ABSTRACT

This paper investigates an information retrieval (IR) based approach for source code plagiarism detection. The method of extensively checking pairwise similarities between documents is not scalable for large collections of source code documents. To make the task of source code plagiarism detection fast and scalable in practice, we propose an IR based approach in which each document is treated as a pseudo-query in order to retrieve a list of potential candidate documents in a decreasing order of their similarity values. A threshold is then applied on the relative similarity decrement ratios to report a set of documents as potential cases of source-code reuse. Instead of treating a source code as an unstructured text document, we explore term extraction from the annotated parse tree of a source code and also make use of field based language model for indexing and retrieval of source code documents. Results confirm that source code parsing plays a vital role in improving the plagiarism prediction accuracy.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval—*Retrieval models, Relevance Feedback, Query formulation*

General Terms

Theory, Experimentation

Keywords

Source Code Plagiarism Detection, Field Search

1. INTRODUCTION

Community question answering (CQA) forums and programming related blogs have made source code widely available to be read, copied and modified. Programmers often tend to re-use source code snippets that are available on the web. The massive amount of programing resources on the web makes it infeasible in practice to perform manual analysis of suspicious source code re-usage. Consequently, there is a need of developing automated methods for detecting source code plagiarism. This is particularly the case for software development companies who want to preserve their intellectual property.

The problem of software plagiarism is challenging because the bag-of-words encoding of source codes in a particular programming language is likely to result in a massive number of *hits* (non-zero similarity values) due to the use of similar programming language specific constructs and keywords. It is therefore highly inefficient to compute pairwise similarity values between source codes in a reasonably large sized collection.

This pairwise computation can be avoided by an information retrieval (IR) based approach. In this approach, first each document is added to an inverted list based indexed organization and then each document in turn can be treated as a pseudo-query to retrieve a ranked list of similar documents from the collection. The plagiarized documents can then be selected from the retrieved list of documents. The research questions in the IR based approach are the following.

- Q1: Does a bag-of-words model (as used in standard IR) suffice or should the source code structure be used in some way to extract more meaningful pieces of information?
- Q2: How to index the source code documents so that a retrieval model can best make use of the indexed terms to retrieve relevant (plagiarized) documents at top ranks?
- Q3: How to represent a source code document as a pseudo-query, i.e. what are most likely representative terms in a source code document?

The rest of the paper describes our investigation on these research questions as a part of our participation in the source code plagiarism detection task SOurce COde reuse (SOCO) in FIRE 2014 [4]. Section 2 discusses the limitations of the existing approaches for software plagiarism detection and motivates the need for an IR approach. In Section 3, we describe our IR based approach to index a large collection of source code documents and retrieve candidate plagiarized documents from this indexed collection. In Section 4, we present the results on the development set of documents and our official results on the test collection. Finally, Section 5 concludes the paper with directions for future work.

2. LIMITATIONS OF EXISTING METHODS

In this section, we first describe why some standard techniques of document similarity estimation may likely fail to work for source codes. We then discuss our proposed method where we attempt to alleviate each of these problems.

2.1 Near Duplicate Document Detection

It is usually the case in software plagiarism that only a part of the source code is copied for reuse in another program. Occurrences of exact duplicates at the level of whole documents is rare. Consequently, the standard techniques of near duplicate document detection from large collections, such as shingling [1], may not be useful for this problem because the Jacard coefficient of the set of shingles for two source codes (a part of one being copy-pasted into the other) is likely to be low.

2.2 Bag-of-words Model

Most existing approaches for source code plagiarism detection take into account the programming language specific features. This is because a bag-of-words encoding of the source codes is likely to result in falsely high similarity values between non-plagiarized documents pairs due to the use of similar programming language specific constructs and keywords.

Programs tend to use a frequent set of variable names, specially for looping constructs, e.g. i, j, k etc., which may also cause false high similarities with a flat bag-of-words representation of documents. Furthermore, programs extensively make use of common library classes, such as the "ArrayList", "HashMap" etc. in Java, which may also contribute to the false matches; for instance two Java programs making use of the standard library "HashMap" may be falsely identified as plagiarized pairs. It is therefore of utmost importance to take into account the structure of a program while computing the similarity. In fact, it has been shown that a control-flow graph based analysis of program pairs produces significantly better results than a simple term frequency based approach [2].

2.3 Exhaustive Pair-wise Similarity

An exhaustive pairwise similarity computation between all documents pairs in a collection is clearly intractable for large sized collections. However, all previously reported approaches of source code plagiarism detection, that we are aware of, use an exhaustive pairwise similarity computation approach [2, 3]. Due to the difficulty in carrying out experimental investigations on large software collections, such approaches are mostly evaluated on a very small collection of source codes, e.g. the evaluation in [2] uses a collection of 56 programs. A standard way to avoid per pair similarity computatiion is to use an inverted list organization of documents to retrieve a candidate list of top most similar documents with respect to a given query. Next, we describe how such an approach can be applied for the software plagiarism detection task.

3. IR APPROACH

In this section, we first describe how an IR based approach is used to obtain a set of candidate plagiarized documents from a ranked list of documents retrieved in response to a pseudo-query constituted from the current document under consideration, and then describe in details how are the documents indexed and how are the pseudo-queries formulated.

3.1 Retrieval with Pseudo-query

To detect all plagiarized document sets in a collection, we propose to treat every document in the collection as a pseudo-query and retrieve a list of top ranked most similar documents in response to the query. However, in contrast to the standard method of result presentation with the help of a ranked list in IR, the objective in the case of plagiarism detection is to obtain a set of documents that are to be predicted as plagiarized.

To get this candidate set of plagiarized documents from the ranked list of (say 1000) retrieved documents, we need to cut-off the ranked list at some point because the ones further down the list are progressively less likely to be relevant (plagiarized). The use of thresholding to obtain a smaller set of candidate documents have been reported in previous works [2, 3].

The cut-off strategy that we use in particular is a thresholding on the relative drops in similarity values of the ranked list. More precisely, we go on accumulating documents from the ranked list of retrieved documents until the relative decease in similarity of the i^{th} ranked document with respect to the $(i-1)^{th}$ one is higher than a pre-defined threshold, say ϵ, as shown in Equation 1. The reasoning behind this is that the first relative drop higher than the threshold most likely indicates the start of non-plagiarized documents. Intuitively speaking, the relative similarity decrement values in contrast to the absolute similarity values are normalized and hence are devoid of any document and query length effects.

$$Plag(Q) = \{D_i : \frac{sim(Q, D_i) - sim(Q, D_{i-1})}{sim(Q, D_{i-1})} \leq \epsilon\} \quad (1)$$

3.2 Document Representation

As discussed in Section 2, the bag-of-words document model representation of a source-code document cannot effectively capture the cases where a part of the source code is copy-pasted into another. To alleviate this problem, a solution is to take into account the code structure of a source code while representing the source document as a vector. Specifically, as a part of document processing for indexing, we used a Java parser[1] to construct an annotated syntax tree (AST) from each source code document in our collection. We then extract terms from specific nodes of the AST.

The words extracted from the list of AST nodes shown in Table 3 are then indexed in stored in separate fields of a Lucene index. A field representation of a document is supposed to better utilize the document structure than a flat view, e.g. a match in the string literal field is treated separately in comparison to a match in the class name field, as a result of which a program using the string constant "HelloWorld" is not considered as plagiarized from a source which defines a class named "HelloWorld".

3.3 Query Representation

Since only a part of the source code is typically copy-pasted into another one, it is not reasonable to use whole source code documents as a pseudo-queries. Instead, we extract a pre-set number of terms from each field of a document

[1] `http://code.google.com/p/javaparser/`

Table 1: Results on the training data.

Parametes				Metric		
AST	Fields	#Qry terms	n-gram	Precision	Recall	F-score
no	no	all	1	0.8000	0.0952	0.1702
no	no	50	1	0.6363	0.4166	0.5035
yes	no	all	1	0.7778	0.0833	0.1505
yes	no	50	1	0.7631	0.3452	0.4754
yes	yes	all	1	**0.8235**	0.1666	0.2772
yes	yes	50	1	0.7894	0.3571	0.4918
no	no	all	2	0.7667	0.2738	0.4035
no	no	50	2	0.7200	0.4285	0.5373
yes	no	all	2	0.7391	0.2023	0.3177
yes	no	50	2	0.5714	0.2857	0.3809
yes	yes	all	2	0.7826	0.2142	0.3364
yes	yes	50	2	0.6842	**0.6190**	**0.6500**

Table 2: SOCO Official Results.

Run Name	Parametes				Metric		
	Parse	Fields	#Qry terms	n-gram	Precision	Recall	F-score
dcu-run1	no	no	50	2	0.432	0.995	0.602
dcu-run2	yes	no	50	2	**0.530**	0.995	**0.692**
dcu-run3	yes	yes	50	2	0.515	**1.000**	0.680

Table 3: Annotated Syntax Tree nodes of a Java program from which terms are extracted during indexing.

Field Name	Field Description
Classes	The names of the classes used in a Java source
Method calls	The method names and actual parameter names and types
String literals	Values of the string constants
Arrays	Names of arrays and dimensions
Method definitions	Names of methods and formal parameter names and types
Assignment statements	Variable names and types
Package imports	Names of imported packages
Comments	

(see Table 3) for constructing a pseudo-query. The term selection function that we use in particular is the language modeling (LM) term score [5], shown in Equation 2.

$$LM(t, f, d) = \lambda \frac{tf(t, f, d)}{len(f, d)} + (1 - \lambda) \frac{cf(t)}{cs} \quad (2)$$

Specifically speaking, in order to formulate a query from a document d, we score each term of each field of d by the function shown in Equation 2 and then select the top most k ones, where k is a parameter. The parameter λ controls the relative importance of the term frequency as against the collection frequency.

4. EXPERIMENTS AND RESULTS

In this section, we report the results of the experiments conducted on the training data and the official results of the SOCO task [4].

4.1 Baselines

As baseline approaches, we submitted two runs in the SOCO task. The first simply uses the standard LM retrieval with a flat bag-of-words representation. Source code documents are treated similar to non-structured text documents and the index does not constitute of separate fields.

As a second baseline approach, we submitted a run where we extract only the terms from the selected nodes of the AST (see Table 3). However, we do not store these terms in separate fields; instead, we use a single field to store them.

In addition to the official submissions, we also investigated other approaches with different parameter settings, e.g. using different number of terms while constructing the pseudo-queries, and unigram and bi-gram (word level) indexing.

4.2 Results on Training Data

The results obtained with different parameter settings are shown in Table 1. The first observation that can be made from Table 1 is that the the use of all terms while constructing the pseudo-query from the documents result in a very low recall. The second important observation is that the method of extracting terms from selected nodes of the AST is not much useful without the document field structures. The third observation is that making use of the word bi-grams for indexing and retrieval tends to improve results for all cases.

4.3 Results on Test Set (Official Results)

The official results of our submitted runs are shown in Table 2. It can be seen that the results on the test set are somewhat different from those on the training one. For instance, the a flat index constituted from the AST terms produces very good results which was not the case for the training set (c.f. Table 1). Flat indexing with no parsing yields considerably worse precision (and hence F-score) in comparison to the parsing based approaches. Surprisingly,

field based LM does not turn out to be more effective than the standard bag-of-words LM.

5. CONCLUSIONS AND FUTURE WORK

This paper described our approach to the problem of source code plagiarism detection. The key idea our approach centers around the hypothesis that the program structure is important for determining source code plagiarisms. Both the development set and the official results empirically confirm this hypothesis. Thus, the answer to research question Q1 (see Section 1) is that parsing the source code helps improve more meaningful pieces of information which can in turn be used to improve the accuracy of plagiarism detection.

Whether a field based retrieval model improves results further or not is inconclusive from the results of the development and the test sets. Research question Q2, where we wanted to explore effective ways of document representation, is yet inconclusive because of the apparent anomalous results obtained on the development and the test sets.

In the third research question, namely Q3 (see Section 1), we wanted to explore effective ways of representing a document as a pseudo-query. The results show that an LM based term selection method of selecting representative terms works significantly better than using all terms for pseudo-query formulation.

In future, we would like to explore the reasons for the apparent anomaly between the development set and the test set results. Using different weights for different fields during the retrieval process is another direction for future research.

Acknowledgments

This research is supported by Science Foundation Ireland (SFI) as a part of the CNGL Centre for Global Intelligent Content at DCU (Grant No: 12/CE/I2267).

6. REFERENCES

[1] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000*, pages 1–10, 2000.

[2] D. Chae, J. Ha, S. Kim, B. Kang, and E. G. Im. Software plagiarism detection: a graph-based approach. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1577–1580.

[3] D.-K. Chae, S.-W. Kim, J. Ha, S.-C. Lee, and G. Woo. Software plagiarism detection via the static api call frequency birthmark. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1639–1643, New York, NY, USA, 2013. ACM.

[4] E. Flores, P. Rosso, L. Moreno, and E. Villatoro-Tello. PAN@FIRE: Overview of SOCO Track on the Detection of SOurce COde Re-use. In *Sixth Forum for Information Retrieval Evaluation (FIRE 2014), Bangalore, India*, 2014.

[5] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, CTIT, AE Enschede, 2000.