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

Debasis Ganguly, Gareth J. F. Jones

CNGL: Centre for Global Intelligent Content
School of Computing, Dublin City University
Dublin 9, Ireland.

December 6, 2014
Introduction

- Question answering forums and programming related blogs have made source code widely available to be read, copied and modified.
- Tendency to copy-paste code. Can result in IP violation.
- Research Challenge: Automatically detect cases of software plagiarism.
- SOCO Task@FIRE 2014: Dataset to evaluate plagiarism detection performance.
We undertake an information retrieval (IR) approach to solve this problem.

**Advantages:**

- No need for exhaustive pairwise comparison between documents.
- Faster than pair-wise exhaustive approaches.
- An ad-hoc query mechanism: a ranked list of candidate documents can be retrieved for a suspected document.
Research Challenges

**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 the most likely representative terms in a source code document?
Limitations of Existing Methods

- Near Duplicate Document Detection by Shingling.
  - Only a part of the source code is copied for reuse.
  - Occurrences of exact duplicates at the level of whole documents is rare.
  - Jacard coefficient of the shingles is expected to be low.

- Bag-of-words Model
  - False high similarity values between non-plagiarized documents pairs.
  - Because of the use of similar programming language specific constructs and keywords.
  - Example: Programs tend to use a frequent set of variable names, specially for looping constructs, e.g. i, j, k etc.
Limitations of Existing Methods (Contd.)

- Exhaustive Pair-wise Similarity
  - Most popular approach that works well in practice.
  - Does not scale well for large collections.
  - Often tested on a small collection, e.g. (Chae et. al., 2013) uses a collection of 56 programs for evaluation.
Source Code Parsing

- Different parts of source codes need to be matched differently.
- Example: A program which defines a class “HelloWorld” vs. a program which defines a string literal “HelloWorld”.
- Parse the source code to build an annotated syntax tree (AST).
- Extract terms from specific nodes of the AST.
### Table: Annotated Syntax Tree nodes of a Java program from which terms are extracted during indexing.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>The names of the classes used in a Java source</td>
</tr>
<tr>
<td>Method calls</td>
<td>The method names and actual parameter names and types</td>
</tr>
<tr>
<td>String literals</td>
<td>Values of the string constants</td>
</tr>
<tr>
<td>Arrays</td>
<td>Names of arrays and dimensions</td>
</tr>
<tr>
<td>Method definitions</td>
<td>Names of methods and formal parameter names and types</td>
</tr>
<tr>
<td>Assignment statements</td>
<td>Variable names and types</td>
</tr>
<tr>
<td>Package imports</td>
<td>Names of imported packages</td>
</tr>
<tr>
<td>Comments</td>
<td></td>
</tr>
</tbody>
</table>
Pseudo-Queries from Source Codes

- It is not reasonable to use whole source code documents as a pseudo-queries: A part of the source code is typically copy-pasted.
- Extract a pre-set number of terms from each field of a document by a term scoring function.
- We use the LM term scoring function.

\[
LM(t, f, d) = \lambda \frac{tf(t, f, d)}{len(f, d)} + (1 - \lambda) \frac{cf(t)}{cs} \tag{1}
\]
Plagiarized Set from Ranked List

Thresholding on the relative drops in similarity values of the ranked list.

\[
Plag(Q) = \{D_i : \frac{sim(Q, D_i) - sim(Q, D_{i-1})}{sim(Q, D_{i-1})} \leq \epsilon \}
\] (2)
Baselines

**Run-1:** Standard LM retrieval:
- Flat bag-of-words representation.
- Source codes treated as non-structured text documents.
- Index does not constitute of separate fields.

**Run-2:** Terms extracted from the selected nodes of the AST. No separate fields are used to store them.
- Other approaches explored (not submitted in the SOCO task):
  - Using different number of terms while constructing the pseudo-queries.
  - Unigram and bi-gram (word level) indexing.
## Training Set Results

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>Fields</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Debasis Ganguly et. al. DCU@FIRE
Test Set (Official SOCO) Results

<table>
<thead>
<tr>
<th>Run</th>
<th>Parse</th>
<th>Fields</th>
<th>#terms</th>
<th>n-gram</th>
<th>Prec.</th>
<th>Rcll</th>
<th>F-sc</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcu-run1</td>
<td>no</td>
<td>no</td>
<td>50</td>
<td>2</td>
<td>0.432</td>
<td>0.995</td>
<td>0.602</td>
</tr>
<tr>
<td>dcu-run2</td>
<td>yes</td>
<td>no</td>
<td>50</td>
<td>2</td>
<td>0.530</td>
<td>0.995</td>
<td>0.692</td>
</tr>
<tr>
<td>dcu-run3</td>
<td>yes</td>
<td>yes</td>
<td>50</td>
<td>2</td>
<td>0.515</td>
<td>1.000</td>
<td>0.680</td>
</tr>
</tbody>
</table>

- Test set results somewhat different from the training set ones.
- A flat index constituted from the AST terms produces very good results (which was not the case for the training set).
- Flat indexing with no parsing → 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.
Conclusions

Q1: Program structure is important for determining source code plagiarism. (Both the development set and the official results empirically confirm this).

Q2: Field based retrieval model improves results further? Inconclusive from the results of the development and the test sets.

Q3: Results show that an LM based term selection method of selecting representative terms works significantly better than using all terms for pseudo-query formulation.