Using Natural Language Parsers in Plagiarism Detection

Maxim Mozgovoy
mmozgo@cs.joensuu.fi

Tuomo Kakkonen
tkakkone@cs.joensuu.fi

Erkki Sutinen
sutinen@cs.joensuu.fi

University of Joensuu
Finland

Abstract

The problem of plagiarism detection system design is a subject of numerous works of the last decades. Various advanced file-file comparison techniques were developed. However, most existing systems, aimed at natural language texts, do not perform any significant preprocessing of the input documents. So in many cases it is possible to hide the presence of plagiarism by utilizing some simple techniques. In this work we show how a natural language parser can be used to fight against basic plagiarism hiding methods.

Index terms: plagiarism detection, natural language parsing, string matching, natural language processing.

1. Introduction

Plagiarism in universities is an important problem, remaining as a topic for scientific works for years. The studies of plagiarism include the understanding of phenomenon itself, developing methods of plagiarism prevention and techniques of plagiarism detection. The later problem turns out to be a technical task in many cases, since plagiarism detection can be effectively done with the help of computer tools.

A plagiarizer, though, can make some efforts to hide plagiarism. For example, in program code files it is possible to rename variables and to change control structures, modifying the initial lexical structure of the program. Several techniques, including tokenization [1] and parameterized matching [2] were developed to fight with such changes. However, similar methods are harder to apply for natural language texts, so usually it turns out to be easier to hide plagiarism in this case.

In this work we show that NLP tools can be used as a rough equivalent of tokenization for natural language texts, overcoming simple plagiarism hiding techniques. Our previous work [3] was dedicated to the problem of rewording sentences. The current work shows how to fight with "split match" problem.

2. "Split Match" Problem

Arguably, the most popular detection scheme in modern plagiarism detection systems is file-file content comparison by means of general string matching algorithms. Generally, the systems try to find the best joint coverage of the two files, and treat the size of this coverage as their similarity ratio. For example, running Karp-Rabin greedy string tiling (RKR-GST) [4] is used in YAP3 [5], JPlag [6], and Plaggie [7] systems. Similar approach is utilized in Sherlock [1] and FPDS [8].

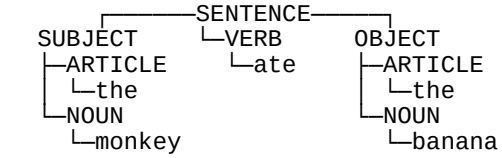
The task of finding optimal joint coverage appears to be NP-complete [5], so heuristic assumptions are used in practical algorithms. For example, nearly all systems use a variation of greedy matching. Another widely used heuristics suggests limiting the minimal length of substrings to be matched. The need of this constraint is caused by the peculiarities of plagiarism detection problem. By including short substrings into the joint coverage, the program provides a lot of false matches that do not indicate plagiarism. Typical copy & paste plagiarism results in duplicating sequences of words, while the presence of the same single words can be an indicator of vocabulary similarity only, not of plagiarism.

Systems like Plaggie [7] and FPDS [8] include such "shortest string length to match" fine-tunable parameters explicitly. Our experiments show that the reasonable size of this parameter is about 10-20 (tokens) in case of program code, and about 4-6 (words) in case of natural language texts.

Unfortunately, the use of this heuristics can cause plagiarism mismatching if a plagiarizer swaps words (this is easy to do in natural language texts, especially in languages such as German, Finnish and Russian that allow freer word order than English). For example, suppose that the original document contains a phrase "light bright sun". A plagiarizer rewords it to "bright light sun". If a system is tuned to match substrings of length 2 (words) and longer only, the plagiarized phrase will not be matched. Keeping in mind the fact that the usual value of "shortest string length to match" constant can be 5-6 words, the intentional word swapping can noticeably affect the detection results.

3. Text Parsing as a Solution

Like computer programs, natural language sentences have syntactic and semantic structure. There are software tools available that can be used to build parse trees for individual sentences. Most automatic English parsers use Chomsky-styled Penn Treebank grammars [9], based on the traditional linguistic approach to the syntax analysis, producing phrase structure-styled analyses. For example, the phrase *the monkey ate the banana* will be represented as



Natural language parsers can recognize noun phrases, homogeneous parts of the sentence, etc. It is clear that word swapping can occur, in particular, in sentences with conjunctions,

such as “and”, “or”, “but”, etc. For example, the phrase “I ate the pizza, the pasta and the donuts” can be reworded as “I ate the pasta, the donuts and the pizza”. Instead of comparing sentences as word strings, we can first analyze them by a parser that recognizes the syntactic structure. These syntactically tagged structures normalize differences between sentences with the same proposition expressed with different word order, thus revealing potential plagiarism.

In contrast to most other parsers based on *probabilistic context-free grammars* (PCFGs), *Stanford Parser* is based on an unlexicalized model [10]. We used version 1.5.1 (30 May 2006) of the system in our experiments. This parser uses a *Cocke-Younger-Kasami* (CYK) [11, 12] search algorithm and can output both dependency and phrase structure analyses [13]. Klein and Manning [10] reported labeled precision and recall figures of 86.9 and 85.7 respectively for this parser. The authors claim that the parser is able to analyze all the sentences in section 23 of the Penn Treebank [9] in a machine with 1GB of memory.

We ran the experiments on the English PCFG grammar and used the dependency output consisting of 48 dependency types. An post-processor tool was implemented in Java that transforms the outputs of Stanford Parser from dependency trees into a format in which the word order has no effect. The format represents the words in the sentence sorted according to their *grammatical relations* (GR) that designate the type of the dependency between the words. The words inside each GR group are sorted in alphabetical order. Figure 1 gives an example of the original Stanford Parser output and transformed format.

```
nsubj(ate-2, I-1)
det(pizza-4, the-3)
dobj(ate-2, pizza-4)
det(pasta-7, the-6)
conj(pizza-4, pasta-7)
cc(pizza-4, and-8)
det(donuts-10, the-9)
conj(pizza-4, donuts-10)

[ate, cc[and], conj[donuts, pasta,
pizza], det[the, the, the],
dobj[pizza], nsubj[I]]
```

Figure 1. Stanford Parser (on the top) and out post-processor (on the bottom) outputs for the sentence: “I ate the pizza, the pasta and the donuts.”

4. Technical Issues

The files generated by natural language parser, are ordinary text documents, and can be used as an input for most general-purpose plagiarism detection system. We have used system [8], earlier developed at our university.

The detection is performed in two phases. First, the parser processes input collection file by file, and generates a collection of parsed files. Second, plagiarism detection system checks the parsed files for similarity. Such flexible scheme allows us to experiment with different parsers, tokenizers, and preprocessors, but in current case has one noticeable drawback. The problem is that the parser destroys the initial word order in every sentence of the input text. Therefore, the plagiarism detection system cannot precisely highlight similar blocks of text in original file pairs. There are two obvious ways to overcome this problem: either the system should be programmed to highlight the whole plagiarized sentences instead of word chains, or the parser should generate some metadata about the parsed files, helping to restore the links between words in original and parsed files. The later is preferable, but requires serious modifications of the parser.

5. Evaluation

Reliable evaluation of a plagiarism detection system is a hard task to perform. Many works use quite informal justifications of the approach used [1, 14]. One of the possible scientific methods includes the use of different plagiarism detection systems as “jury” to evaluate the examined system. Though the separate “opinions” of other systems cannot work as reliable indicators of quality of the system being evaluated, the collaborative “voting” determines the subset of plagiarized files more reliably. This method was used to evaluate the system [8].

Unfortunately, this approach is hard to follow with natural language texts, since the only system we know that utilizes natural language parsing is our recent project [3], that is based on the same system [8], paired with a parser of the Russian language. Therefore, the results were analyzed manually.

Our positive experience with [3] encouraged to use short news messages as an input collection. Such a selection is based on the fact that quite often different agencies provide information about the same event. Furthermore, agencies often cite one another, increasing the number of possible duplications (sometimes reworded). We do not expect plagiarism in this case.

For the evaluation of the system, a collection of 128 messages was obtained from the website of BBC NEWS (http://news.bbc.co.uk). Each message falls into one of the following categories: *Business*, *Europe*, *Science/Nature*, and *Technology*. The median size of each message (after removing all formatting) is about 2 KB.

We have also prepared several files with intentional plagiarism, performed using copy & paste with subsequent change of word and phrase order. It should be mentioned that news is hard to plagiarize with such a method, since the reels are laconic, and do not contain enough adjectives or phrases to swap. However, in free-form essays we used there are more possibilities for such “swap-powered” plagiarism.

The system [8] without parser found 11 pairs of messages containing vast quotations from each other, and 3 pairs of messages informing about the same event. The similarity ratios of later pairs are 5%-33%. The similarity ratios of plagiarized free-form essays were estimated as 10%-30%. The inclusion of parser increases similarity ratios for overlapping news messages by 7%-13%. This observation indicates that the new system has an overall tendency to assign higher similarity grades to the same file pairs. As a result, two more file pairs of similarity 5% and 7% appeared in the resulting log, but they were not considered as similar by human graders. In practice, it is possible to get rid of incorrectly matched pairs by raising a similarity threshold for final file pair list. For the plagiarized free-form essays the similarity ratios have increased significantly — to 50%-80%. The results are also noticeably affected by the value of “shortest string length to match” constant. The smaller is constant, the less effect has the use of the parser. Large constant values cause higher

probability to mismatch "swap-powered" plagiarism, detectable by means of parser.

6. Discussion

Our experiments show that the use of natural language parser to find swapped words and phrases can be effective for intentional plagiarism, but usually impractical for casually similar documents. If two files use the same source or cite each other, the probability of finding intentional swaps is low. On the contrary, plagiarizers have strict motivation to hide copy & paste plagiarism, so swaps are much more likely to occur.

Since the existence of swaps is a good indication of plagiarism, one of the possibilities for future research can be comparison of file pair similarity without and with parsing. The high difference in numbers can indicate the presence of intentional word swaps, and, therefore, of plagiarism.

The use of smaller "shortest string length to match" constant can effectively fight against swaps as well, but it also significantly increases the possibility of false matches. Furthermore, larger values of this constant make detection algorithm work faster [8].

References

[1] M.S. Joy, M. Luck, "Plagiarism in Programming Assignments", *IEEE Transactions on Education*, vol. 42(2), 1999, pp. 129-133.

[2] B.S. Baker, "Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance", *SIAM Journal on Computing*, vol. 26(5), 1997, pp. 1343-1362.

[3] M. Mozgovoy, V. Tusov, V. Klyuev, "The Use of Machine Semantic Analysis in Plagiarism Detection", *Proceedings of the 9th International Conference on Humans and Computers*, Japan, 2006, p. 72-77.

[4] M.J. Wise, "Running Karp-Rabin Matching and Greedy String Tiling", *Technical Report #463*, Basser Department of Computer Science, University of Sydney, 1993.

[5] M.J. Wise, "YAP3: Improved Detection of Similarities in Computer Program and Other Texts", *Proceedings of SIGCSE '96*, 1996, pp. 130-134.

[6] L. Prechelt, G. Malpohl, and M. Philippsen, "JPlag: Finding Plagiarisms among a Set of Programs", *Technical report*, Fakultät für Informatik, Universität Karlsruhe, Germany, 2000.

[7] A. Ahtiainen, S. Surakka, M. Rahikainen, "Plaggie: GNU-Licensed Source Code Plagiarism Detection Engine for Java Exercises", *Proceedings of the 6th Baltic Sea Conference on Computing Education Research*, 2006, pp. 141-142.

[8] M. Mozgovoy, K. Fredriksson, D. White, M. Joy, and E. Sutinen, "Fast Plagiarism Detection System", *Lecture Notes in Computer Science*, vol. 3772, 2005, pp. 267-270.

[9] M.P. Marcus, B. Santorini, M.A. Marcinkiewicz, "Bulding a Large Annotated Corpus of English: the Penn Treebank", *Computational Linguistics*, vol. 19, 1993, pp. 313-330.

[10] D. Klein, C. Manning, "Accurate Unlexicalized Parsing", *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, 2003, pp. 423-430.

[11] T. Kasami, "An Efficient Recognition and Syntax-analysis Algorithm for Context-free Languages", *Scientific Report AFCRL-65-758*, Air Force Cambridge Research Lab, Bedford, Massachusetts, USA, 1965.

[12] D. Younger, "Recognition and Parsing of Context-free Languages in Time n^3", *Information and Control*, vol. 10(2), 1967, pp. 189–208.

[13] M-C. de Marneffe, B. MacCartney, C. Manning, "Generating Typed Dependency Parses from Phrase Structure Parses", *Proceedings of the 5th International Conference on Language Resources and Evaluation*, 2006.

[14] B. Belkhouche, A. Nix, J. Hassell, "Plagiarism Detection in Software Designs", *Proceedings of the 42nd Annual Southeast Regional Conference*, 2004, pp. 207-211.