A plagiarism detection procedure in three steps: selection, matches and "squares" *

Chiara Basile
Dip. Matematica
Univ. Bologna
P.zza di Porta S. Donato 5,
40126, Bologna, Italy
basile@dm.unibo.it

Dario Benedetto
Dip. Matematica
Sapienza, Univ. Roma
P.le Aldo Moro 2,
00185, Roma, Italy
benedetto@mat.uniroma1.it

Emanuele Caglioti
Dip. Matematica
Sapienza, Univ. Roma
P.le Aldo Moro 2,
00185, Roma, Italy
caglioti@mat.uniroma1.it

Giampaolo Cristadoro
Dip. Matematica
Univ. Bologna
P.zza di Porta S. Donato 5,
40126, Bologna, Italy
cristadoro@dm.unibo.it

Mirko Degli Esposti
Dip. Matematica
Univ. Bologna
P.zza di Porta S. Donato 5,
40126, Bologna, Italy
desposti@dm.unibo.it

Abstract: We present a detailed description of an algorithm tailored to detect external plagiarism in PAN-09 competition. The algorithm is divided into three steps: a first reduction of the size of the problem by a selection of ten suspicious plagiarists using a n-gram distance on properly recoded texts. A search for matches after T9-like recoding. A "joining algorithm" that merges selected matches and is able to detect obfuscated plagiarism. The results are briefly discussed.
Keywords: n-grams, plagiarism, coding, string matching

1 Introduction

In this short paper we aim to describe our approach to automatic plagiarism detection. In particular, we discuss how we were able to borrow and adapt some techniques and ideas recently developed by some of us for different, but related, problems such as Authorship Attribution (AA)(Benedetto, Caglioti, and Loreto, 2002; Basile et al., 2008). While some of the authors gained over the last years certain expertise in the field of AA, it is the first time we face plagiarism detection. The algorithm we are going to describe has been tailored on the *"1st International Competition on Plagiarism Detection"* (Stein et al., 2009) and does not pretend to be optimal for generic situations. Indeed we joined the competition while being completely unaware of the relevant literature, and thus the main aim of this paper is to participate to a detailed comparison of the different approaches to the contest that, we hope, will permit to enlarge the applicability of the methods and generate a profound integration and combination of different ideas.

* We thank the organizers of PAN-09 competition for the stimulating challenges and C. Cattuto and V. Loreto for bringing the contest to our attention.

2 The problem and the datasets

The contest was divided into two different challenges: external and internal plagiarism. We concentrated on the external plagiarism only. For the sake of completeness we recall the main goal (see (PAN-09-Organizers, 2009) for more details):

..given a set of suspicious documents and a set of source documents the task is to find all text passages in the suspicious documents which have been plagiarized and the corresponding text passages in the source documents.

The organizers provided a training corpus, composed of 7214 source documents and 7214 suspicious documents, each with an associated XML containing the information about plagiarized passages. A first statistical analysis shows that text lengths vary from few hundreds to 2.5 million characters while the total number of plagiarized passages is 37046. Moreover, exactly half of the suspicious texts contain no plagiarism and about 25% of the source documents are not used to plagiarize any suspicious document. The length of the plagiarized passages has a very peculiar distribution, see Figure 1: there are no passages with length in the window 6000-12000 characters, and even for long texts there is no

plagiarism longer than 30000 characters. A remarkable fact is that about 13% of the plagiarized passages consist of *translated* plagiarism.

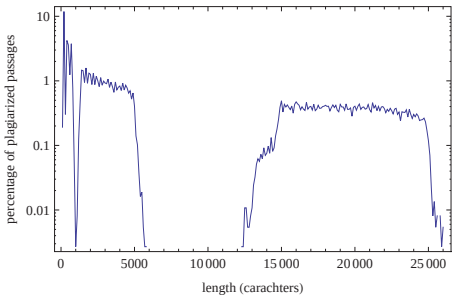

Figure 1: Distribution of plagiarized passage lengths in the training corpus.

Similarly, the competition corpus is composed of 7215 source documents and 7214 suspicious documents (obviously without any associated XML files). The length statistics are very close to those for the training corpus, see Figure 2.

The overall score is then calculated as the ratio between F-measure and granularity over the whole set of detected chars (see (Stein et al., 2009) for more details).

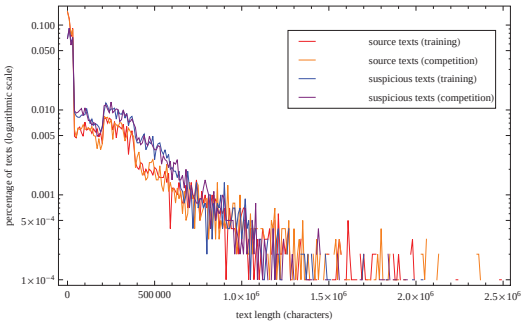

Figure 2: Text length distribution for the training corpus and for the competition corpus.

3 The method

3.1 A first selection

For each suspicious document, we first identify a small subset of source documents for a second and deeper (computationally demanding) analysis.

We start by calculating a suitable distance between each suspicious and source text. Then, for each suspicious text the first 10 source neighbors ordered according to this distance are kept for further analysis.

In order to bring computational time to a practical scale the texts were first transformed into sequences of word lengths, so that for example the sentence `To be, or not to be: that is the question` becomes simply `2223224238`. All word lengths greater than 9 were cut to 9, so that the new alphabet consists of the nine symbols $\{1,\dots,9\}$. These coded versions of the texts are on average 82.5% shorter than the original ones, and so computation times are greatly reduced.

The distance between each suspicious and source text has been computed by comparing in a suitable way the frequency vectors of the 8-grams of the coded versions. This n-gram distance used here has been proved successful in Authorship Attribution problems (Basile et al., 2008). To be more precise, after a first experiment based on bigram frequencies presented in 1976 by Bennett (Bennett, 1976), Kešelj et al. published in 2003 a paper in which n-gram frequencies were used to define a similarity distance between texts (V. Kešelj and Thomas, 2003). The distance introduced in (Basile et al., 2008) and used here should be considered as a natural development of the previous ones: once the value of n has been fixed, usually between 4 and 8, n-gram frequencies are calculated for a given text x. If we denote by ω an arbitrary n-gram, by $f_x(\omega)$ the relative frequency with which ω appears in the text x and by $D_n(x)$ the so called *n-gram dictionary* of x, that is, the set of all n-grams which have nonzero frequency in x, for any pair of texts x and y, we can define:

$$d_n(x,y) := \frac{1}{|D_n(x)|+|D_n(y)|} \sum_{\omega\in D_n(x)\cup D_n(y)} \left(\frac{f_y(\omega)-f_x(\omega)}{f_y(\omega)+f_x(\omega)}\right)$$

This is exactly the distance that has been used in (Basile et al., 2008), together with a suitable voting procedure, for approaching a two-class classification problem.

The parameter $n = 8$ for the $n-$gram distance was chosen here as a compromise between a good recall (the fraction of plagiarized characters coming from the first 10 nearest neighbors of each suspicious text is 81%) and acceptable computation times (about 2.3 days for the whole corpus). Since it was impossible to do repeated computations on the whole corpus, the optimization was done using a small subset of 160 suspicious texts and 300 source texts, suitably selected by imposing total and plagiarism length distributions comparable to those of the whole corpus.

Note that a recall of 81% is a very good result for the 8-gram distance, since the method just described has basically no hope

to recognize the 13% of translated plagiarism. Therefore, at least on the small subset of the training corpus, only about 6% of the (non translated) plagiarized passages are lost in this phase.

3.2 T9 and matches

After identifying the 10 "most probable plagiarist sources" we now perform a more detailed analysis to detect the plagiarized passages. The idea is to look for common subsequences (*matches*) longer than a fixed threshold. To this goal we need to recover some of the information lost on the first passage by first rewriting the text in the original alphabet and then using a different (and less "lossy") coding. We perform a T9-like coding: this system is typically used for assisted writing on most mobile phones. The idea is to translate three or four different letters into the same character, for example {`a,b,c`} $\mapsto 2$, {`d,e,f`} $\mapsto 3$ and so on. The symbol 0 is used for newline and blank space, 1 for all symbols other than these two and the letters of the alphabet. The new alphabet for the coded texts is therefore made up of 10 symbols: $\{0, 1, 2, \ldots, 9\}$. Note that the use of T9 "compression", which could seem strange at a first sight, can be justified by observing that a "long" T9 sequence (10-15 characters) has in most cases an "almost unique" translation in a sentence which makes sense in the original language.

The "true" matches between a suspicious and a source document are now found: from any possible starting position in the suspicious document the longest match in the source document is calculated (possibly more than one with the same length). If the length is larger than a fixed threshold and the match is not a submatch of a previously detected one, it is stored in a list.

Here, we take advantage of the choice of the T9 encoding, which uses ten symbols: for any starting position in the source document, the algorithm stores the last previous position of the same string of length 7, and for any possible string of length 7 it is memorized, in a vector of size 10^7, the last occurrence in the source file. With respect to other methods (suffix trees or sorting, for istance), in this way we can search the maximum match in the source document avoiding to compare the smaller ones.

Running this part of the algorithm on a common PC for the whole corpus of 7214 texts took about 40 hours. The threshold for the match length was fixed arbitrarily to 15 for texts shorter than 500000 characters, to 25 for longer texts.

3.3 Looking for "squares"

The previous phase gives a long list of matches for each suspicious-source pair of documents. Since the plagiarized passages had undergone various levels of obfuscation, typically the matches are "close" to each other in the suspicious texts but taken from not necessarily subsequent places in the source texts. By representing the pair of texts in a bidimensional plot, with the suspicious text on the x axis and the source text on the y axis, each match of length l, starting at x in the suspicious document and at y in the source document, draws a line from (x, y) to $(x + l, y + l)$. The result is often something similar to Figure 3: there are some random (short) matches all around the plane but there are places where matches accumulate, forming lines or something similar to a square. Non-obfuscated plagiarism corresponds to lines, i.e. a single long match or many short close matches which are in succession both in the suspicious and in the source texts, whereas intuitively obfuscated plagiarism corresponds to "squares": here the matching sequences are in a different order in the source and suspicious documents.

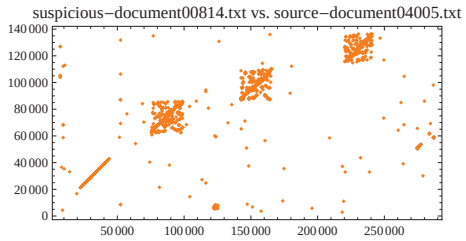

Figure 3: Orange points correspond to the position of matching chars (measured in number of chars from the beginning) between a suspicious and a source document (see top of the plot) of the training corpus. A "square" of matches corresponds to an obfuscated plagiarism.

Figure 4 is an example of what can happen when there is no plagiarism: matches are uniformly spread around the plane and do not accumulate anywhere. Obviously these are just two of the many possible settings: longer texts or the presence of "noise" (e.g. long sequences of blanks, tables of numbers...) can give rise to a much higher density of

matches, substantially increasing the difficulties in identifying the correct plagiarized passages.

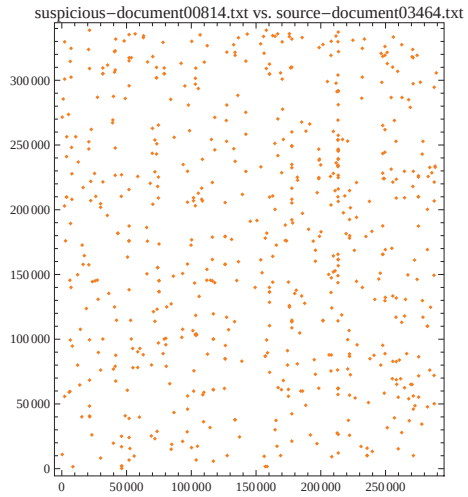

Figure 4: Orange points correspond to the position of matching chars (measured in number of chars from the beginning) between a suspicious and a source document (see top of the plot) of the training corpus. No plagiarism is present in this case.

In order to provide a single quadruple (x, y, l_x, l_y) of starting points and lengths for each detected plagiarized passage we need to implement an algorithm that joins the "cloud" of matches of each "square".

Note that the algorithm that performs this task needs to be scalable with plagiarism lengths, which can vary from few hundreds, up to tens of thousands characters.

The algorithm used here *joins two matches* if the following conditions hold simultaneously:

1. matches are subsequent in the x coordinate;
2. the distance between the projections of the matches on the x axis is greater than or equal to zero (no superimposed plagiarism) but shorter than or equal to the l_x of the longest of the two sequences, scaled by a certain δ_x;
3. the distance between the projection of the matches on the y axis is greater than or equal to zero but shorter than or equal to the l_y of the longer of the two sequences, scaled by a certain δ_y.

Merging now repeatedly the segments which are superimposed either in x or in y, we obtain some quadruples which correspond roughly to the "diagonals" of the "squares" in Figure 3. We finally run the algorithm once again using smaller parameters δ'_x and δ'_y in order to reduce the granularity from 2 to approximately the optimal value of 1. Figure 5 shows the result of the algorithm for the couple of texts of Figure 3 (blue), and Figure 6 shows the very good superimposition with the actual plagiarized passages (black), as derived from the XML file.

The procedure just described has been developed and tuned for the competition in a restricted time schedule, but it would be quite interesting to compare the efficiency of this procedure to the ones that can be achieved by using standard clustering algorithms. We plan to do this in the future.

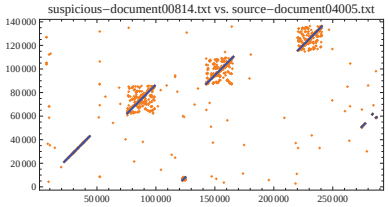

Figure 5: Detected plagiarism for the pair of texts of the training corpus indicated at the top of the plot. Single matches in orange, joined matches in blue.

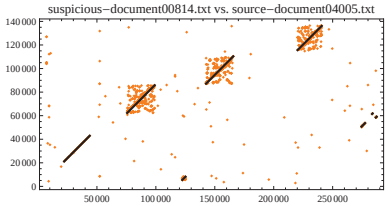

Figure 6: Plagiarized passages for the pair of texts of the training corpus indicated at the top of the plot. Single matches in orange, actual plagiarism in black. Note the perfect superimposition between the blue lines in figure 5 and the black lines here.

3.4 Tuning the parameters

The "joining algorithm" described above depends on 4 parameters: δ_x and δ_y for the first joining phase, and the rescaled δ'_x and δ'_y for the second joining phase. Our choice of the actual value in use has been dictated essentially by the lack of time and no rigorous and efficient optimization has been performed. Driven by very few trials and with some heuristic, we decided to use the following values: $\delta_x = \delta_y = 3$ and $\delta'_x = \delta'_y = 0.5$.

It is important to remark that different choices of the δ values yield to different detection results. For example, increasing their values typically results in a larger recall and in a better granularity, but also in a smaller precision. A further analysis of these dependencies could provide (in future developments) a controllable way of modifying the precision, the recall and the granularity, depending on the plagiarism detection task into consideration. A promising strategy that we plan to explore in the future consists in a dynamical tuning of these parameters according, for example, to the density of matches or to the lengths of the two texts into consideration.

The match-joining algorithm was developed using *Mathematica*© 7, and it ran on a common PC for about 20 hours on the whole data set of the competition.

4 Results and comments

The algorithm described gave the following results on the competition corpus (Stein et al., 2009):

- Precision: 0.6727
- Recall: 0.6272
- F-measure: 0.6491
- Granularity: 1.0745
- Overall score: 0.6041

The overall score is the third best result after 0.6093 and 0.6957 of the first two. We stress that the overall score drops considerably starting from the fourth position (0.3045), the fifth (0.1885), and so on. Moreover, while the winner has better results in all precision, recall and granularity, our precision and recall are better than the second, while granularity is worse.

The competition had a very tight schedule, therefore many improvements are possible. In particular, it may be that the match-joining problem can be advantageously formulated in the framework of Hidden Markov Models. Also, it would be worth to see how standard clustering algorithms perform in this case.

We are eager to compare techniques and ideas with the other participants of the contest.

References

Basile, C., D. Benedetto, E. Caglioti, and M. Degli Esposti. 2008. An example of mathematical authorship attribution. *Journal of Mathematical Physics*, 49:125211–125230.

Benedetto, D., E. Caglioti, and V. Loreto. 2002. Language Trees and Zipping. *Physical Review Letters*, 88(4):048702–1, 048702–4.

Bennett, W.R. 1976. Prentice-Hall, Englewood Cliffs, NJ.

PAN-09-Organizers. 2009. Proceedings of PAN-09.

Stein, B., M. Potthast, A. Eiselt, P. Rosso, and A. Barrón-Cedeño. 2009. http://www.webis.de/pan-09/competition.php.

V. Kešelj, F. Peng, N. Cercone and C. Thomas. 2003. n-gram-based author profiles for authorship attribution.