ENCOPLOT: Experiments and Results in Automatic Plagiarism Detection

Cristian Grozea, Ph.D.
cristian.grozea@first.fraunhofer.de

Fraunhofer Institute FIRST – Berlin

September 24, 2010

The Speaker

- "Jäger der Plagiatoren"
- "Kommissar Algorithmus"
- Bester "Spürhund" für Plagiate

- formerly lecturer at the Faculty of Mathematics and Computer Science, University of Bucharest
- currently researcher at Fraunhofer Institute FIRST in Berlin

Outlook

- My early work (2004, while teaching at University of Bucharest) – information based plagiarism detection.
- The method Encoplot – the first international competition on automatic plagiarism detection
- Determining the direction of the plagiarism
- Human or Machine?

The public search engines

- "google" it!
- it's a manual method
- only retrieves the indexed documents
- TurnItIn.com?

Possibly the best plagiarism detection

Many ways to see copying/plagiarism between two texts:

- common substrings
- redundancy
- **common information**
- deficiency of the novel information

Not practical!

Information quantity is uncomputable.
No reasonable approximation for information exists.
Approximation through compression.

1st International Competition on Plagiarism Detection

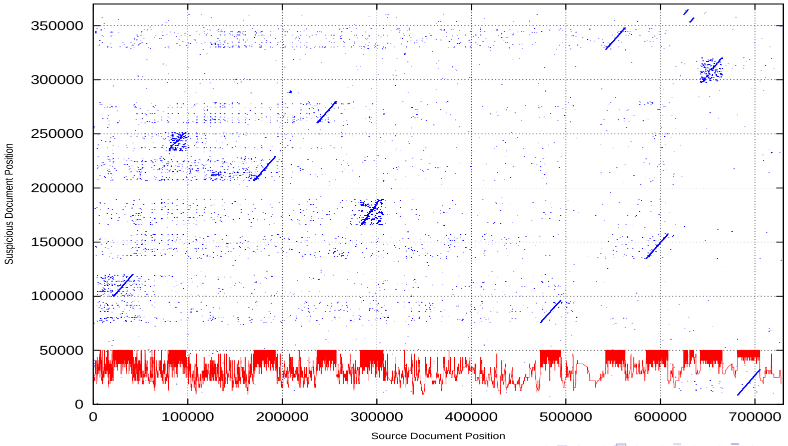
- Training dataset, plagiarism annotated
- Test dataset, unannotated, used for evaluation
- each 7000 source documents and 7000 suspicious documents
- Automatic plagiarism and obfuscation: reorder paragraphs, change and insert or delete words
- Two tasks: intrinsic plagiarism (e.g. by style), external plagiarism (find the source in a given list and indicate what passages are copied from where)

Results

Ranked 1st in the 1st International Competition on Plagiarism Detection; Ranked 4th in the 2nd International Competition on Plagiarism Detection (intrinsic+ext).

Rank	Overall score	F-measure	Precision	Recall	Granularity	Participant
1	0.6957	0.6976	0.7418	0.6585	1.0027	C. Grozea Fraunhofer FIRST, Germany
2	0.6093	0.6192	0.5573	0.6967	1.0164	J. Kasprzak, M. Brandejs, and M. Kipa Masaryk University, Czech Republic
3	0.6041	0.6491	0.6727	0.6272	1.0745	C. Basile(a), D. Benedetto(b), E. Caglioti (a)Universit di Bologna and (b)Universit L

Encoplot: source 3094 vs suspicious 9



Encoplot Features

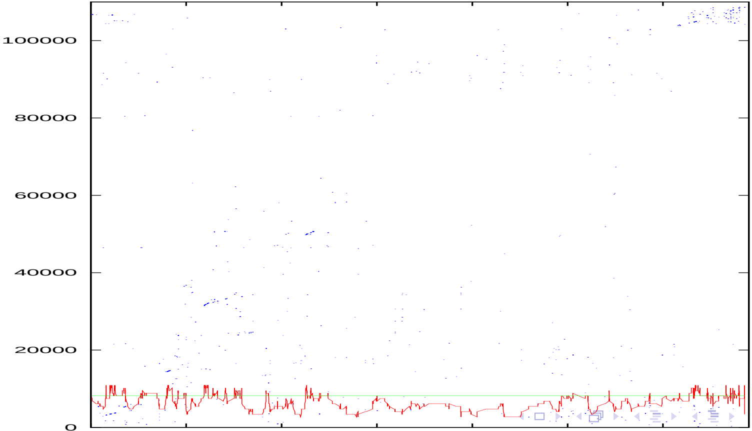
- Guaranteed linear time (Dotplot is quadratic).
- Field-agnostic, possible to use in computational biology as well, for example.
- Extremely fast highly optimized implementation available (for N up to 16, on 64 bit CPUs).

Example 2

- Article submitted for review to the very person it has been plagiarized from!
- Plagiarism *without* idea theft.
- Difficult, as a result of rephrasing and changing, it is far from verbatim copying.

Encoplot

Source versus Copy (texts)



Copied passage 1

/root/copyspot/test/srcpart, Top line: 1

nnot be used directly to obtain the new SVM state. The problem lies in the changing composition of the sets S and R with the change of Δs and Δαc in Eq. (4). To handle this problem, the main strategy of the algorithm is to identify the largest increase Δαc such that some point migrates between the sets S and R. Four cases must be considered to account for such structural changes: 1. Some αi in S reaches a bound (an upper or a lower one). Let Compute the sets S I+ = {i ∈ S : βi > }

S I− = {i ∈ S : βi < − }. S The examples in set I+ have positive sensitivity with respect to the weight of the current example; that is, their weight would increase by taking the step Δαc .2 These

/root/copyspot/test/dstpart, Top line: 1

not be used directly to obtain the new state such that all the samples satisfy the KKT conditions except that the restriction (9) does not need to hold for the weights of the enlarged jc -th two-class training samples. The problem lies in the changing composition of the sets SS , SR and SE following the increment of αc . To handle this problem, the main strategy of the max for each incremental adjustment algorithm is to compute the maximal increment Δαc (see Fig. 3) such that a certain sample migrates among the sets SS , SR and SE . Three cases must be considered to account for such structural changes: 1. Some αi in SS reaches a bound (an upper or a lower bound). Compute the sets: S S c c c I+S = {i ∈ SS : βi > 0}, I−S = {i ∈ SS : βi < 0}, where the samples with βi = 0 are ignored due to their insensitivity to Δαc . Thus the maximum possible weight updates are S C − αi , if i ∈ I+S max Δαi = (19) S −αi , if i ∈ I−S

S and the maximal possible Δαc S before a certain sample in SS moves to SR or SE is max Δα S Δαc S = mini∈I SS ∪I SS βic .

tried in Google

e.g. No results found for "the problem lies in the changing composition of the sets".

Copied passage 2

/root/copyspot/test/srcpart, Top line: 1

```
h training data is provided one example at a
 time, as opposed to the batch mode in which
 all examples are available at once (e.g.
 Robbins and Munro (1951); Murata (1992);
 Saad (1998); Bishop (1995); Orr and M¨ller
 (1998); LeCun et al. (1998); Murata et al.
 (2002)). u Online learning is advantageous
 when dealing with (a) very large or (b)
 non-stationary data. In the case of
 non-stationary data, batch algorithms will
 generally fail if ambiguous information,
 e.g. di□erent distributions varying over
 time, is present and is erroneously
 integrated by the batch algorithm (cf.
 Murata (1992); Murata et al. (2002)). Many
 problems
c 2006 Pavel Laskov, Christian Gehl, Stefan
 Kr¨ger and Klaus-Robert M¨ller. u u

□¨ ¨ Laskov, Gehl, Kruger and Muller

of high interest in machine learning can be
 naturally viewed as online ones. An
 important practical advantage of online
 algorithms is that they allow to
 incorporate additional training data, when
 it is available, without re-training from
 scrat
```

/root/copyspot/test/dstpart, Top line: 1

```
, training data is usually provided one
 example at a time, and this is the so
 called online scenario. We again use flight
 delays forecasts as an example. The given
 flight delay data streams are
 non-stationary, meaning that data
 distributions vary over time. Batch
 algorithms will generally fail if such
 ambiguous information is present and is
 erroneously integrated by the batch
 algorithm; but incremental learning
 algorithms are more capable in this case,
 because the advantage of incremental
 learning algorithms is that they allow the
 incorporation of additional training data
 without re-training from scrat
```

Copied passage 3

/root/copyspot/test/srcpart, Top line: 1

0 contains the inner product values for all 1 ≤ i, j ≤ n. The matrix K is obtained from the kernel matrix by incorporating the labels: K = K0 (yy T).

The operator denotes the element-wise matrix product, and a vector y denotes labels as an n × 1 vector. Using this notation, the SVM training problem can be formulated as

/root/copyspot/test/dstpart, Top line: 1

h contains the inner product values K(xi , xk) for all 1 ≤ i, k ≤ l. Then the matrix Q is obtained from the kernel matrix by incorporating the labels: Q = (yT y) H, where the operator denotes the element-wise matrix product, and the vector y denotes labels as an 1 × l vector. After introducing these notations, the dual function can be formulated as

Copied passage 4

/root/copyspot/test/srcpart, Top line: 1

zero, which would allow an example to be brought into S. The problem remains – since Δμ is free as opposed to non-negative Δαc – to determine the direction in which the components of gr are pushed by changes in μ. This can be done by first solving (25) for Δμ, which yields the dependence of Δgr on Δgc : Δgr = − yr Δgc . yc

Since Δgc must be non-negative (gradient of the current example is negative and should be brought to zero if possible), the direction of

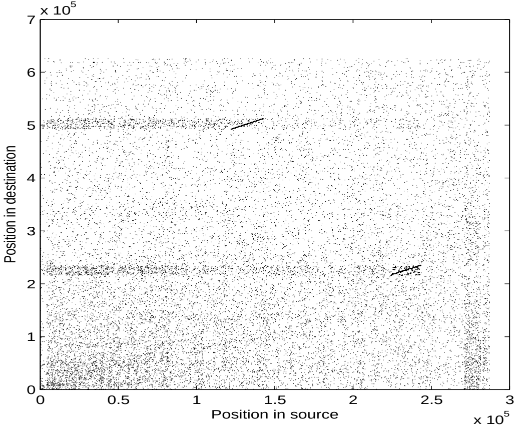
/root/copyspot/test/dstpart, Top line: 1

zero, which would allow a j j sample to be brought into SSc . Consequently, SSc is no longer empty. The remaining problem is to determine the direction of change in bjc because the sign of Δbjc is free as opposed to non-negative Δαc . This can be done

Performance

- *libmindy* able to compute kernel matrix with 49 millions elements in 12 hours on an 8-core machine.
- encoplot + heuristic clustering of dots able to do detailed analysis (passages matching) for 350000 document pairs in less than 8 hours.
- We didn't have much time, still we spent more time thinking than building and running programs.

Determining the direction of plagiarism



- Who's the Thief? Automatic Detection of the Direction of Plagiarism, C.Grozea and M.Popescu, CICLING 2010 , LNCS 6008, DOI 10.1007/978-3-642-12116-6, 2010
- ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection, C.Grozea, C.Gehl, and M.Popescu – In Proceedings of the 3rd PAN Workshop. Uncovering Plagiarism, Authorship and Social Software Misuse, San Sebastian, Spain, 2009. Universidad Politecnica de Valencia 2009
- Encoplot – Performance in the Second International Plagiarism Detection Challenge, C. Grozea and M. Popescu, Lab Report for PAN at CLEF 2010
- Plagiarism Detection with State of the Art Compression Programs, C.Grozea Report CDMTCS-247, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, 2004.

Thank you! (time to take questions)

What is and what is not plagiarism

- Copying of text - unless it's quoting - is plagiarism.
  Easy to detect
  - can be detected at the text level
- Copying ideas is also plagiarism.
  Not so easy to detect
  - can be seen at semantic level
- Self-plagiarism: Copying text from your own previous papers.
  Unclear
  - it is not considered plagiarism by some

students papers become the property of Turnitin.com

Students Settle With TurnItIn In Copyright Case 2009
Fair Use Affirmed In Turnitin Case 2009
Students Sue Anti-Plagiarism Service 2007
Students Protest Turnitin.com 2006
Online Plagiarist Sues University 2004
Student Fights University Over Plagiarism-Detector 2004
Turnitin.com - Placebo for Plagiarism or Worse? 2002

Approximating the ideal plagiarism detection

Requirements for a practical compressor function C:
$C(x|x) = O(1)$ where x is any text. Equiv. $C(xx) = C(x)$.
$C(f(x)|x) = O(1)$ where f is any computable transformation. Equiv. $C(xf(x)) = C(x)$.

TokenCompress in Chen, Francia, Li, McKinnon, Seker (2003) - Shared Information and Program Plagiarism Detection
BZIP2 in Grozea (2004) - Plagiarism Detection with State of the Art Compression Programs

Approximating the approximation

Compression can be slow.
The best compression is most always the slowest.
Detecting that two documents have too much in common does not mean that we can see what they have in common.
Text compression is mostly based on coding the text repetitions.
ZIP, BZIP2, to some extent even PPM.
Focusing on "repetitions" as a form of redundancy could solve both: detecting most forms of redundancy between documents and identifying the passages in correspondence.

N-Gram Coincidence Plot

Algorithm

Input: Sequences A and B to compare

Output: list (x,y) of positions in A, respectively B, where there is exactly the same N-gram

Steps

1. Extract the N-grams from A and B
2. Sort these two lists of N-grams
3. Compare these lists in a modified mergesort algorithm.

Whenever the two smallest N-grams are the equal, output the position in A and the one in B.

Small example

A=abcabd
B=xabdy

N=2

Encoplot pairs	Dotplot pairs
1 2 ab	1 2 ab
	4 2 ab
5 4 bd	5 4 bd

N=3

Encoplot pairs	Dotplot pairs
4 2 abd	4 2 abd

Encoplot vs Dotplot Analysis

Question: what is the price paid for speed?

Encoplot matches the first N-gram in text *A* with the first identical N-gram in the text *B*, the second occurence with the second occurence and so on.

Encoplot may break sequences on N-grams that are duplicated in one of the texts. A sequence too fragmented may no longer lead to the recognition of a suspicious match.

Being duplicated means their informational content is reduced (e.g. typical formulations such as "despite this, we are").

Only the parts that are rather unique in each of the text are guaranteed to be put in correspondence. Hopefully these correspond to high information substrings, "signatures" that really identify the text.

Optimizations

- No N-gram extraction has actually being performed, just sorting of the indexes with the N-grams as keys.
- The sorting method is radix sort (linear time) further optimized for N-grams, by incrementally updating the symbol occurence counters (one symbol in, one symbol out, at each of the N steps).
- A 16-gram fits into an elementary (almost native) gcc type: __uint128_t = 2 64 bit registers

Challenge approach

No stemming (looked like it brings only 1% improval of the performance).
Used 16-grams, character based, as opposed to word based - good for avoiding to treat common formulations as significant.

- Computation of a kernel matrix (49 million pairs) using a linear kernel over binary representation of 16-grams (ignoring frequency in document), normalized.
- Selection of the pruning: best worked ranking using the kernel the suspicious documents for each source document. [Reasons].
- Kept 50 "most suspicious" for each source.

Challenge approach – continued

- For each (source, suspicious) pair in the about 350,000 kept, compute the encoplot and apply a heuristic to isolate the clusters (diagonals), in linear time.
- Filter once more the list of detections, in order to only keep the very convincing matches (long, still holding after whitespace elimination, high matching score). This increases the precision (less false positives) with the price of decreasing the recall (more false negatives).

Fast Radix Sort for N-Grams

```
for(i,NN)ix[i]=i;
//radix sort, the input is x,
// the output rank is ix
for(k,RANGE)counters[k]=0;
for(i,NN)counters[*(x+i)]++;
for(j,DEPTH){
   int ofs=j;//low endian
   t_int sp=0;
   for(k,RANGE){
      startpos[k]=sp;
      sp+=counters[k];
   }
   for(i,NN){
      unsigned char c=x[ofs+ix[i]];
      ox[startpos[c]++]=ix[i];
   }
   memcpy(ix,ox,NN*sizeof(ix[0]));
   //update counters
   if(j<DEPTH-1){
      counters[*pout++]--;
      counters[*pin++]++;
   }
}
```

What is AIT?

- **An information theory of individual, finite objects (strings)**
  00000000000000000000
  01010001001001100 1
  Σ a set of symbols, Σ^* the set of all words over Σ, including λ, the zero-length word. Usually $\Sigma = \{0, 1\}$ the binary alphabet.
- **Built around descriptive complexity (here Kolmogorov), compressibility**

$$K(y) = \min_{x \in \Sigma^*} \{| x |; U(x) = y\}$$

where U is a universal "machine" – e.g. universal Turing machine, or general purpose programming language interpreter and $| x |$ is the length of the string x

Properties of the complexity

- It does not depend on the choice of the universal machine.

$$K_U(x) = K_V(x) + O(1)$$

  where O(1) is a constant term, not depending on x.
- **The complexity is uncomputable.** It can be approximated as the limit of upper bounds.
  Decompression is algorithmic, ultimate compression is not.
- The complexity of a string is at most its length.

$$K(x) \leq \mid x \mid + O(1)$$

- Most strings are incompressible. For example, out of all length n strings, only a fraction of $2^{-\frac{n}{2}}$ are compressible to half their size.

Other basic results

- The computable functions can only add limited (by a constant) information to that of their arguments

$$K(f(x)) \leq K(x) + K(f) + O(1)$$

- Injective computable functions preserve the information

$$K(f(x)) = K(x) + O(1)$$

when f is injective.

- Complexity is not only uncomputable but fully equivalent to the halting problem (Chaitin et al. 1995)

Prefix-free sets

- **Prefix:** x is a prefix of y, if there is a z such that $xz = y$ (concatenation).
- **Prefix-free set:** no element of the set is a prefix of another element.
- **Kraft's inequality:** If S is a prefix-free set, then

$$\sum_{x \in S} 2^{-|x|} \leq 1$$

Prefix-free sets (2)

- **Kraft-Chaitin Theorem** If a set of natural numbers $L = \{l_i\}_i$ satisfies the Kraft's inequality

$$\sum_{l_i \in L} 2^{-l_i} \leq 1$$

then there exists a prefix-free set $S = \{s_i\}_i$ such that $\mid s_i \mid = l_i$.

- "Kraft-Chaitin Inequality Revisited", C.Calude and C.Grozea, 1996
- "Free-Extendible Prefix-Free Sets and an Extension of the Kraft-Chaitin Theorem", C.Grozea, 2000

Prefix complexity – Chaitin

- **Chaitin complexity**

$$H(y) = \min_{x \in \Sigma^*} \{ | x |; U(x, \lambda) = y \}$$

  where U is a universal prefix Turing machine (with prefix-free domain).

- **Conditional complexity**

$$H(y|z) = \min_{x \in \Sigma^*} \{ | x |; U(x, z) = y \}$$

  x is the "program", z is the input.

- **Complexity of a pair of strings**

$$H(x, y) = H(< x, y >)$$

  where $< \cdot, \cdot >: {\Sigma^*}^2 \to \Sigma^*$ is a fixed injective computable

Prefix complexity – Chaitin (2)

- $H(x|x) \leq 0 + O(1)$
- $H(x|y) \leq H(x) + O(1)$
- $H(x,y) \leq H(x) + H(y|x) + O(1)$
- $H(xy) \leq H(x,y) + O(1)$
- **Common information**

$$I(x,y) = H(x) + H(y) - H(x,y)$$

AIT

Quantize the information using an Information Theory framework - e.g. AIT

H = Chaitin complexity

- Independent x and y

$$H(xy) \approx H(x) + H(y)$$

$$H(y|x) \approx H(y)$$

- Dependent x and y

$$H(xy) \ll H(x) + H(y)$$

Chen, Francia, Li, McKinnon, Seker (2003) - Shared Information and Program Plagiarism Detection