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).

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

C.Grozea
ENCOPLOT – Tool for Automatic Plagiarism Detection
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.
Source versus Copy (texts)
not 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 $S_{I^+} = \{i \in S : \beta_i > \}$  

$S_{I^-} = \{i \in S : \beta_i < -\}$. 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 $\Delta\alpha c$. 2. These
e.g. No results found for "the problem lies in the changing composition of the sets".
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 Müller (1998); LeCun et al. (1998); Murata et al. (2002)). 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. different distributions varying over time, is present and is erroneously integrated by the batch algorithm (cf. Murata (1992); Murata et al. (2002)). Many problems 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 scratch.
Copied passage 3

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

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

\[
\frac{1}{1} C.Grozea
\]
zero, which would allow an example to be brought into $S$. The problem remains – since $\Delta \mu$ is free as opposed to non-negative $\Delta \alpha_c$ – to determine the direction in which the components of $g_r$ are pushed by changes in $\mu$. This can be done by first solving (25) for $\Delta \mu$, which yields the dependence of $\Delta g_r$ on $\Delta g_c$: $\Delta g_r = -y r \Delta g_c$. $y_c$

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


Encoplot – Performance in the Second International Plagiarism Detection Challenge, C. Grozea and M. Popescu, Lab Report for PAN at CLEF 2010

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 = abcabbd
B = xabdy

<table>
<thead>
<tr>
<th>Encoplot pairs</th>
<th>Dotplot pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 ab</td>
<td>1 2 ab</td>
</tr>
<tr>
<td>4 2 ab</td>
<td></td>
</tr>
<tr>
<td>5 4 bd</td>
<td>5 4 bd</td>
</tr>
</tbody>
</table>

N = 2

<table>
<thead>
<tr>
<th>Encoplot pairs</th>
<th>Dotplot pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 2 abd</td>
<td>4 2 abd</td>
</tr>
</tbody>
</table>
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 occurrence with the second occurrence 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 occurrence 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% improvement 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

```c
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++];
    }
}
```

C.Grozea
What is AIT?

- An information theory of individual, finite objects (strings)
- Σ a set of symbols, Σ* the set of all words over Σ, including λ, the zero-length word. Usually Σ = {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.
  \[ K(x) \leq |x| + O(1) \]

- Most strings are incompressible. For example, out of all length \( n \) strings, only a fraction of \( 2^{-n/2} \) 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 $|s_i| = l_i$.

- “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 \(<\cdot, \cdot>: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*\) is a fixed injective computable function.
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**

\[
I(x, y) = H(x) + H(y) - H(x, y)
\]
Quantize the information using an Information Theory framework - e.g. AIT

$H = \text{Chaitin complexity}$

- 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