Antiplagiarism system for R language

Overview, conclusions and ideas

Maciej Bartoszuk
16 czerwca 2016

Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej
Table of contents

1. Introduction
2. Antiplagiarism - general information
3. Input sequences
4. Preliminary choice of function pairs
5. Comparison algorithms
6. Experimental results
7. Presentation of antiplagiarism results to an end user
8. Future work
Introduction
R language

R [1] language is used in many fields:

- Statistical computations
- Data analysis
- Data mining
- Machine learning
- Bioinformatics
Where the antiplagiarism system is needed?

- Teaching students – increasing quality of education process
- Research of CRAN packages dependencies
- Code cloning/reusing in big IT projects – increasing quality of software
Existing antiplagiarism systems

- JPLAG [2, 3], based on tokens
- MOSS [4], based on tokens, $q$-grams
- GPLAG [5, 6, 7], based on program dependency graph

All of them are dedicated to another languages, like C++ or Haskell. Experimental results showed they do not deal with R very well.
Facts and myths about antiplagiarism systems
Facts and myths about antiplagiarism systems – Myth #1

One could think it works in fire-and-forget (odpal i zapomnij) way: the source codes are inputed, and a system detects plagiarisms with 100% certainty, and after that it sends an e-mail to a dean and remove students from USOS or set zero points for a task.
Facts and myths about antiplagiarism systems – Myth #2

One could think also that, it is so smart that if two students wrote the same algorithm with the same code, but one initialize a variable with zero (and it is correct), while the second one with one (incorrect), the system would recognize it and consider it as not a plagiarism.
Facts and myths about antiplagiarism systems – Fact #1

No system executes the code which is examined. It is inconvenient and dangerous.
Some people suggest that to compare results returned by functions. Strange (and the same) results of two functions should trigger an alert. It contradicts Fact #1 and also would be inconvenient for an end user. End user wants fire-and-forget, remember?
There are no fire-and-forget systems. There are many reasons for that:

- A description of a task can impose a solution,
- In every submission different degree of similarity is suspect
- Some information about students’ relationships is sometimes helpful
- Some information about circumstances in which a task was written is sometimes helpful
- Ability to solve a task of each student is also helpful
Facts and myths about antiplagiarism systems – Fact #3

So what can we offer?

- a sorted list of pairs sorted by some similarity measure
- displaying two functions next to each other
- displaying two functions after some normalization (without comments, the same indentation style)
Antiplagiarism - general information
Assumptions can change as work progresses, but for today:

- We calculate similarity between two *functions* $f_i$ and $f_j$ in R. We do not consider scripts, subset of function nor group of functions.
- We assume that there is only R code in function and there are no C++ calls.
Figure 1: Overview
Typical attacks from plagiarists

Easy:

- Add/remove comments
- Change names of variables
- Change “<-” into “=” or “->”

Moderate:

- Change order of lines of code
- Add/remove line(s) of code
- Expand/shrink of function calls, e.g.:

```r
1 x[order(unlist(lapply(x, f)))]
```

and

```r
1 y <- unlist(lapply(x, f))
2 o <- order(y)
3 x[o]
```
Typical attacks from plagiarists

Hard:

- Change loop into its equivalent form (for into while, but also into lapply), e.g.:

```r
1  y <- numeric(n)
2  k <- 1
3  for (i in x)
4  {
5      y[k] <- sqrt(i)
6      k <- k+1
7  }

and

```r
1  y <- unlist(lapply(x, function(element){ return(sqrt(element))}))
```
or even

```r
1  y <- sqrt(x)
```
Method $\mu$ should be:

- Reflexive: $\mu(f_i, f_i) = 1$,
- There is no need to be symmetric $\mu(f_i, f_j) = \mu(f_j, f_i)$ (!),
- Transitivity also can be discussed.
Why method should not be symmetric?

Consider example, where $f_1$:

1. $s \leftarrow 0$
2. for $(i \text{ in } x)\{s \leftarrow s + i\}$

and $f_2$:

1. $s \leftarrow 0$
2. for $(i \text{ in } x)\{s \leftarrow s + i\}$
3. $m \leftarrow 0$
4. for $(i \text{ in } x)\{m \leftarrow m \ast i\}$

We are interested in method which returns $\mu(f_1, f_2) = 1$ and $\mu(f_2, f_1) = 0.5$. 
Transitivity discussion

Consider example, where $f_1$:

```r
s <- 0
for (i in x){s <- s + i}
```

and $f_2$:

```r
s <- 0
for (i in x){s <- s + i}
model <- glm( class ~ age + hiEduc, family=binomial)
```

and $f_3$:

```r
model <- glm( class ~ age + hiEduc, family=binomial)
```

We are interested in method which returns $\mu(f_1, f_2) \approx 0.5$ and $\mu(f_2, f_3) \approx 0.5$, but $\mu(f_1, f_3) = 0$. But maybe in later work we should use transitive closure and find clusters?
Some method $\mu_1$:

- Transform source code to one of the three input sequences
- Use one of the three comparison algorithms

It gives us some number of methods.
Input sequences
Possible sequences

- Letters
Pros and cons of letters

Advantages:

- Easy to implement,
- Deals fairly well with easy attacks, such as changing names of variables

Drawbacks:

- It does not "understand" code, so it cannot take advantage of parse information, such as loops, function calls, variable assignments etc.
- Cannot deal with more sophisticated attacks.
Possible sequences

- Letters
- Tokens
Based on [2, 3].

Let’s begin with an example. For function $f$:

```r
f <- function(x) {
  stopifnot(is.numeric(x))
  y <- sum(x)
  y
}
```

we obtain such tokens:

```
1 SYMBOL, LEFT_ASSIGN, FUNCTION, '(' , SYMBOL_FORMALS, ')' ,
2 '{' ,
3 SYMBOL_FUNCTION_CALL, '(' , SYMBOL_FUNCTION_CALL, '(' , SYMBOL, ')' , ')' ,
4 SYMBOL, LEFT_ASSIGN, SYMBOL_FUNCTION_CALL, '(' , SYMBOL, ')' ,
5 SYMBOL,
6 '}'
```

As we can see, we try to obtain some more general symbols from source code than string of letters.

We try to find big “tiles” of matching tokens in both functions.
Pros and cons of tokens

Advantages:

- Invulnerable for changing names of variables
- Based on some parse data,
- Deals with swapping big fragments of code.

Drawbacks:

- It is easy to get false positive, because some different fragments of code can result in the same tokens sequence, e.g. call two different functions with the same number of arguments (and every argument is a variable).
- Does not deal with swapping small fragments of code.
- Generally does not deal with expanding/shrinking function calls.
Possible sequences

- Letters
- Tokens
- Function calls counts
Pros and cons of function calls counts

Advantages:

• Easy to implement
• Surprisingly very effective
• Hard to deceive by all types of attacks

Drawbacks:

• In theory many false positives can be obtained, but experimental results does not confirm this concern,
• Sometimes there are “synonyms” for functions, such as \texttt{nrow()} (number of rows), \texttt{ncol()} (number of columns) and \texttt{dim()} (number of rows and columns together),
• Plagiarist can create aliases for functions, e.g. \texttt{l <- lapply} and use \texttt{l} instead of
Possible sequences

- Letters
- Tokens
- Function calls counts
- Program Dependence Graph
Based on [5, 6, 7].

Algorithm:

- Create program dependence graph for every function $f_i$,
- Compare how similar two program dependence graphs are
What is a program dependency graph?

Program dependency graph consists of:

- Control dependency graph
- Data dependency graph
sum <- function(x) {
  s <- 0
  m <- 1
  for (i in x) {
    s <- s + i
    m <- m * i
  }
  if (s < 0) {
    s <- -s
    print("Negative s")
  }
  if (m < 0) {
    m <- -m
    print("Negative m")
  }
  return(s)
}
Data dependency graph

Data dependency edges are blue.

```
1 sum <- function(x)
2 {
3   a <- 5
4   b <- 6
5   for (i in x)
6     {
7       c <- a + b - i
8     }
9 }
```
Pros and cons of program dependency graph

Advantages:

• This method uses the most information from parse tree, so theoretically has the biggest potential,
• Immune to changing names of variables, swapping lines of code, expanding/shrinking of function calls,
• Deals with changing loop types.

Drawbacks:

• Difficult to implement,
• Algorithms comparing two graphs can be expensive,
• False positives and negatives are possible.
Preliminary choice of function pairs
• Extract and save some features from examined functions
Ideas

- Extract and save some features from examined functions
- Find a way to get two functions, where features are the same, but these functions are of different length (the difference is large)
Ideas

- Extract and save some features from examined functions
- Find a way to get two functions, where features are the same, but these functions are of different length (the difference is large)
- MOSS examines $q$-grams
Ideas

- Extract and save some features from examined functions
- Find a way to get two functions, where features are the same, but these functions are of different length (the difference is large)
- MOSS examines $q$-grams
  - Some idea is using metric trees, but the triangle inequality has to be fulfilled,
Ideas

- Extract and save some features from examined functions
- Find a way to get two functions, where features are the same, but these functions are of different length (the difference is large)
- MOSS examines $q$-grams
- Some idea is using metric trees, but the triangle inequality has to be fulfilled,
- **Does the data can be stored in database and is adding new data possible?**
LDA [8] fulfills all (or most) requirements.

- „successor” of Latent Semantic Analysis (LSA) method, which is also called Latent Semantic Indexing (LSI)
LDA [8] fulfills all (or most) requirements.

- „successor” of Latent Semantic Analysis (LSA) method, which is also called Latent Semantic Indexing (LSI)
- In the literature every word of source code is used,
LDA [8] fulfills all (or most) requirements.

- "successor" of Latent Semantic Analysis (LSA) method, which is also called Latent Semantic Indexing (LSI)
- In the literature every word of source code is used,
- In this work functions names and tokens are used
LDA [8] fulfills all (or most) requirements.

- „successor” of Latent Semantic Analysis (LSA) method, which is also called Latent Semantic Indexing (LSI)
- In the literature every word of source code is used,
- In this work functions names and tokens are used
- Function names are additionally splitted by dot (.), dash (-) or underscore mark (_), e.g., is.numeric or stri_locate_all.
Latent Dirichlet Allocation
• $\alpha = 0.05$ – when $\alpha$ is large, nearly every document will be composed of every topic in significant amounts. In contrast when $\alpha$ is small, each document will be composed of only a few topics in significant amounts,
• $\alpha = 0.05$ – when $\alpha$ is large, nearly every document will be composed of every topic in significant amounts. In contrast when $\alpha$ is small, each document will be composed of only a few topics in significant amounts,

• $\beta$ – estimated by the algorithm, larger values of $\beta$ favor a greater number of words per topic, while smaller values of $\beta$ favor fewer words per topic.
Latent Dirichlet Allocation – technical details

- $\alpha = 0.05$ – when $\alpha$ is large, nearly every document will be composed of every topic in significant amounts. In contrast when $\alpha$ is small, each document will be composed of only a few topics in significant amounts,
- $\beta$ – estimated by the algorithm, larger values of $\beta$ favor a greater number of words per topic, while smaller values of $\beta$ favor fewer words per topic.
- Gibbs sampling
• $\alpha = 0.05$ – when $\alpha$ is large, nearly every document will be composed of every topic in significant amounts. In contrast when $\alpha$ is small, each document will be composed of only a few topics in significant amounts,

• $\beta$ – estimated by the algorithm, larger values of $\beta$ favor a greater number of words per topic, while smaller values of $\beta$ favor fewer words per topic.

• Gibbs sampling

• $k = \max \left( \frac{\text{NumberOfFunctions}}{10}, 2 \right)$ – topic count
Latent Dirichlet Allocation – technical details

- $\theta$ – some parameter,
- for a pair of functions $(f_i, f_j)$:
  - get sets of topics $T_i$ and $T_j$, where assignments of topic are above $\theta$,
  - $m_i$ – topic with maximum assignment in $T_i$, the same for $m_j$ and $T_j$,
  - $T_{ij} = T_i \cap T_j$,
  - the pair $(f_i, f_j)$ should be compared if $m_i \in T_{ij}$ or $m_j \in T_{ij}$. 
Comparison algorithms
Comparison algorithms

- Edit distance
Comparison algorithms

• Edit distance
• Generalized longest common subsequence
Comparison algorithms

- Edit distance
- Generalized longest common subsequence
- Smith-Waterman Algorithm [9]
Smith-Waterman Algorithm

- A CACAC T A
- A G CACAC A
## Smith-Waterman Algorithm

The Smith-Waterman algorithm is used to find the globally optimal local alignment between two sequences. It is a dynamic programming algorithm that allows for gaps in the alignment.

### Example:

Consider the sequences:

- **Query**: ACTA
- **Template**: AGGA

The scoring matrix `H` for these sequences is:

\[
H = \begin{pmatrix}
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \\
G & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
C & 0 & 1 & 3 & 2 & 3 & 2 & 3 & 2 \\
A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 4 \\
G & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 \\
C & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 8 \\
A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 \\
\end{pmatrix}
\]

The algorithm fills in the matrix `H` by comparing each character in the query sequence with each character in the template sequence, considering the scoring matrix and the gap penalties. The optimal alignment is then traced back through the matrix to find the best match.
Smith-Waterman Algorithm

T =

\begin{pmatrix}
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
G & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
C & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
A & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
C & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
A & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
A & 0 & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\
\end{pmatrix}
Comparison algorithms

- Edit distance,
- Generalized longest common subsequence,
- Smith-Waterman Algorithm,
- $q$-grams distance,
Comparison algorithms

- Edit distance,
- Generalized longest common subsequence,
- Smith-Waterman Algorithm,
- $q$-grams distance,
- McGregor algorithm (Most common graph problem),
Comparison algorithms

- Edit distance,
- Generalized longest common subsequence,
- Smith-Waterman Algorithm,
- $q$-grams distance,
- McGregor algorithm (Most common graph problem),
- Graph kernel (graphs, labels)
Weisfeiler-Lehman subtree graph kernel

Described in [10].

Given labeled graphs $G$ and $G'$

1st iteration
Result of steps 1 and 2: multiset-label determination and sorting

End of the 1st iteration
Feature vector representations of $G$ and $G'$

$$\phi^{(1)}_{WL_{subtree}}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$\phi^{(1)}_{WL_{subtree}}(G') = (1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1)$$

Counts of original node labels

Counts of compressed node labels

$$k^{(1)}_{WL_{subtree}}(G, G') = \langle \phi^{(1)}_{WL_{subtree}}(G), \phi^{(1)}_{WL_{subtree}}(G') \rangle = 11.$$
Symmetric and not symmetric versions of methods

- All of aforementioned methods can be formulated in symmetric version 
  \( \mu_k(f_i, f_j) = \mu_k(f_j, f_i) \) and not symmetric 
  \( \mu_k(f_i, f_j) \neq \mu_k(f_j, f_i) \)

- In classic scenario, when group of functions is submitted and the system is supposed to find similar functions only in the set, not symmetric versions are used,

- symmetric versions are used when distances between functions are desired to create a metric space (needed for metric trees, described further)
T-norms

- Not symmetric methods seems to be more flexible,
- unfortunately, the most common scenario is when user/tutor wants only these pairs, in which both functions are both similar to each other,
- what is more, one pair of functions should be displayed only once on website (legibility, readability)
- so the question is how to aggregate two values $\mu_k(f_i, f_j)$ and $\mu_k(f_j, f_i)$ into one?
The answer is a t-norm. A t-norm is a function $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$ which satisfies the following properties:

- **Commutativity:** $T(a, b) = T(b, a)$
- **Monotonicity:** $T(a, b) \leq T(c, d)$ if $a \leq c$ and $b \leq d$
- **Associativity:** $T(a, T(b, c)) = T(T(a, b), c)$
- **The number 1 acts as identity element:** $T(a, 1) = a$

Among exemplary t-norms we find:

- minimum $T(a, b) = \min(a, b)$,
- product $T(a, b) = a \cdot b$,
- Łukasiewicz t-norm: $T(a, b) = \max(0, a + b - 1)$.

So big value of any t-norm assure that both functions are similar to each other, while small value means that at least one function is not so similar to the second.
Experimental results
Creating learning set

- We obtained homeworks from students
- We manually found plagiarisms
- We also created artificial plagiarism functions for found pairs of plagiarism (ca. 30,000 unique pairs of functions)
Creating testing set

- We obtained ca. 400 pairs from a Similar.Rexamine.com,
- 5 grades of plagiarism can be chosen: totally different, dissimilar, hard to say, similar and definitely similar,
- We classified options similar and definitely similar as a plagiarism class and the rest as not plagiarism.
Notation

- **True Positives (TP):** Plagiarism from learning set detected by our system
- **False Positives (FP):** Detected pair which is not plagiarism
- **False Negatives (FN):** Plagiarism pair not detected by our system
- **True Negatives (TN):** Pair which is not plagiarism and is not detected by our system
Observations

- 2% of all function pairs are plagiarisms,
- System which always returns “no plagiarism” has 98% accuracy,
- It is totally useless,
- We have to use another methods of assessment, like recall (how many of all plagiarism pairs are detected) and precision (how many of returned pairs are actually plagiarisms)
Methods of assessment

- Error rate is \( \frac{FP+FN}{TP+FP+FN+TN} \),
- Precision is \( \frac{TP}{TP+FP} \),
- Recall is \( \frac{TP}{TP+FN} \),
- Accuracy is \( \frac{TP+TN}{TP+FP+FN+TN} \).
Our approach is to consider data from 4 methods as a following data frame:

<table>
<thead>
<tr>
<th>f1</th>
<th>f2</th>
<th>method1</th>
<th>method2</th>
<th>method3</th>
<th>method4</th>
<th>plagiarism</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectorSum</td>
<td>listSummation</td>
<td>0.6</td>
<td>0.7</td>
<td>0.9</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>vectorSum</td>
<td>regexpTagger</td>
<td>0.2</td>
<td>0.4</td>
<td>0.1</td>
<td>0.2</td>
<td>0</td>
</tr>
</tbody>
</table>

We build statistical model for such data (of course we do not consider functions’ names). We chose a random forest for now, but future work on choosing appropriate statistical system is planned.
Possible methods

1. program dependence graph – maximum common subgraph isomorphism,
2. letters – edit distance,
3. f.calls – edit distance,
4. tokens – edit distance,
5. letters – generalized longest common sequence,
6. f.calls – generalized longest common sequence,
7. tokens – generalized longest common sequence,
8. letters – 1-grams distance,
9. f.calls – 1-grams distance,
10. tokens – 1-grams distance,
11. letters – 2-grams distance,
12. f.calls – 2-grams distance,
13. tokens – 2-grams distance,
14. letters – 3-grams distance,
15. f.calls – 3-grams distance,
16. tokens – 3-grams distance,
17. letters – 4-grams distance,
18. f.calls – 4-grams distance,
19. tokens – 4-grams distance
<table>
<thead>
<tr>
<th>Features</th>
<th>Acc.</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F-meas.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assym.</td>
<td>0.9893</td>
<td>0.814</td>
<td>0.704</td>
<td>0.755</td>
</tr>
<tr>
<td></td>
<td>Sym.</td>
<td>0.9881</td>
<td>0.887</td>
<td>0.557</td>
<td>0.684</td>
</tr>
<tr>
<td>1, 7</td>
<td>Assym.</td>
<td>0.9928</td>
<td>0.831</td>
<td>0.863</td>
<td>0.847 tokens – GLCS</td>
</tr>
<tr>
<td></td>
<td>Sym.</td>
<td>0.9925</td>
<td>0.849</td>
<td>0.825</td>
<td>0.837</td>
</tr>
<tr>
<td>1, 2, 7, 9</td>
<td>Assym.</td>
<td>0.9963</td>
<td>0.922</td>
<td>0.924</td>
<td>0.923 standard approach</td>
</tr>
<tr>
<td></td>
<td>Sym.</td>
<td>0.9957</td>
<td>0.912</td>
<td>0.900</td>
<td>0.906</td>
</tr>
<tr>
<td>1, 3, 4, 7, 8, 9, 10</td>
<td>Assym.</td>
<td>0.9977</td>
<td>0.967</td>
<td>0.928</td>
<td>0.947 statistical methods</td>
</tr>
<tr>
<td>12, 15, 16, 18, 19</td>
<td>Assym.</td>
<td>0.9971</td>
<td>0.955</td>
<td>0.923</td>
<td>0.939</td>
</tr>
<tr>
<td>1, 2, 3, 4, 6, 7, 9, 13, 15, 16, 18, 19</td>
<td>Assym.</td>
<td>0.9977</td>
<td>0.969</td>
<td>0.933</td>
<td>0.951 common sense</td>
</tr>
<tr>
<td></td>
<td>Sym.</td>
<td>0.9975</td>
<td>0.954</td>
<td>0.943</td>
<td>0.949</td>
</tr>
</tbody>
</table>
Dataset

- Dataset from the Example,
- the number of unique observations equals to \( m = 30628 \),
- the benchmark data set is of the following form:

<table>
<thead>
<tr>
<th>( j )</th>
<th>( x_1^{(j)} )</th>
<th>( x_2^{(j)} )</th>
<th>( x_3^{(j)} )</th>
<th>( x_4^{(j)} )</th>
<th>( y^{(j)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.82</td>
<td>0.73</td>
<td>0.63</td>
<td>0.92</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.58</td>
<td>0.41</td>
<td>0.84</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>0.15</td>
<td>0.25</td>
<td>0.38</td>
<td>0.48</td>
<td>0.50</td>
</tr>
<tr>
<td>4</td>
<td>0.37</td>
<td>0.26</td>
<td>0.40</td>
<td>0.39</td>
<td>0.25</td>
</tr>
<tr>
<td>5</td>
<td>0.17</td>
<td>0.02</td>
<td>0.11</td>
<td>0.12</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>0.22</td>
<td>0.13</td>
<td>0.46</td>
<td>0.28</td>
<td>0.25</td>
</tr>
<tr>
<td>7</td>
<td>0.69</td>
<td>0.90</td>
<td>0.72</td>
<td>0.80</td>
<td>0.75</td>
</tr>
<tr>
<td>8</td>
<td>0.87</td>
<td>0.70</td>
<td>0.83</td>
<td>0.92</td>
<td>1.00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Figure 2: Exemplary B-spline basis functions $N_{j-p,p}^t(x)$ as a function of $x$, $j = 1, \ldots, p + k + 1$; $t$ is a vector of equidistant knots, $k$ is the number of the internal knots, while $p$ is the polynomial degree.
Optimization details

- Rewrite the above equation in terms of a bi-level minimization procedure,
- the inner-level part, for a fixed $w$, optimizes for $c$ and in fact can be written in the form of a standard quadratic programming task with linear constraints

$$\min \sum_{l=1}^{m} \left( \sum_{i=1}^{n} w_i \left( \sum_{j=1}^{n} c_j^{(i)} N_{j-p,p}^{t} \left( x_i^{(l)} \right) \right) - y^{(l)} \right)^2 + \lambda_w \sum_{i=1}^{n} w_i^2 \quad \text{w.r.t. } w, c$$

- the outer-level component, optimizing for $w$, can be solved via some non-linear solver – we propose to rely on the CMA-ES [11, 12] algorithm and logarithmic barrier functions for the constraints on $w$. 
**Performance**

**Table 1**: Performance of the fitted models (accuracy, precision, recall, \( F \)-measures, squared \( L_2 \) error). The proposed method is based on \( \lambda_w = 33, w_1 = 0.35, w_2 = 0.15, w_3 = 0.15, w_4 = 0.35, p = 3, k = 1 \) for optimizing \( F \)-measure (a) and \( \lambda_w = 30, w_1 = 0.30, w_2 = 0.16, w_3 = 0.15, w_4 = 0.39, p = 1, k = 4 \) for optimizing \( \hat{\sigma}_2^2 \) (b).

<table>
<thead>
<tr>
<th>method</th>
<th>accuracy</th>
<th>precision</th>
<th>recall</th>
<th>( F )</th>
<th>( \hat{\sigma}_2^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method (a)</td>
<td>0.997</td>
<td>0.921</td>
<td>0.933</td>
<td>0.927</td>
<td>106.62</td>
</tr>
<tr>
<td>Proposed method (b)</td>
<td>0.997</td>
<td>0.900</td>
<td>0.920</td>
<td>0.910</td>
<td>95.85</td>
</tr>
<tr>
<td>Linear regression</td>
<td>0.995</td>
<td>0.810</td>
<td>0.969</td>
<td>0.883</td>
<td>103.53</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>0.997</td>
<td>0.885</td>
<td>0.960</td>
<td>0.921</td>
<td>—</td>
</tr>
<tr>
<td>Random forest</td>
<td>0.998</td>
<td>0.927</td>
<td>0.956</td>
<td>0.941</td>
<td>—</td>
</tr>
</tbody>
</table>
## Table 2: Performance measures as functions of different weighting vectors; $p = 3$, $k = 1$, $\lambda_w = 0$, with and without idempotization.

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>$F$</th>
<th>$\varphi_2^2$</th>
<th>Idempot.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.992</td>
<td>0.848</td>
<td>0.693</td>
<td>0.763</td>
<td>186.30</td>
<td>Yes</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.995</td>
<td>0.927</td>
<td>0.787</td>
<td>0.851</td>
<td>208.74</td>
<td>Yes</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.994</td>
<td>0.803</td>
<td>0.853</td>
<td>0.828</td>
<td>316.04</td>
<td>Yes</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.996</td>
<td>0.904</td>
<td>0.840</td>
<td>0.871</td>
<td>136.67</td>
<td>Yes</td>
</tr>
<tr>
<td>0.27</td>
<td>0.06</td>
<td>0.38</td>
<td>0.29</td>
<td>0.996</td>
<td>0.952</td>
<td>0.800</td>
<td>0.870</td>
<td>137.34</td>
<td>No</td>
</tr>
<tr>
<td>0.41</td>
<td>0.12</td>
<td>0.07</td>
<td>0.40</td>
<td>0.997</td>
<td>0.919</td>
<td>0.907</td>
<td>0.913</td>
<td>107.69</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Figure 3: Best B-splines of different degrees fit to the training sample.

(a) $p = 1, k = 4$

(b) $p = 3, k = 1$
Figure 4: $F$-measure and squared error as a function of the $\lambda_w$ regularization coefficient.
Presentation of antiplagiarism results to an end user
When we have the results, some tools which make possible to have a big picture of the results are needed,

one of them is clustering. A spectral clustering is used to group functions into "families". It can be useful to get a clique of cheating students or to realize that all students have very similar solutions in one task for some reason,

other methods of visualization are planned, e.g. a graph of distances,

single-linkage clustering
Future work
Possible enhancements

• Inserting a code of called functions to a code of examined function,

• Displaying pairs of function in the three-dimensional space,

• Improve function comparing PDGs
Example

Katarzyna Z:

```r
odl <- function(space, x1, x2){
  sqrt(((space[,1] - space[,2])^2 +
       (space[,2] - space[,2])^2))
}

findClosestPoints <- function(space){
  stopifnot(is.matrix(space), is.numeric(space),
            ncol(space) == 2, nrow(space) > 0, all(is.finite(space)))
  r <- nrow(space)
  if(r==1) return(Inf)
  if(r==2) return(odl(space, 1, 2))
  if(r==3) return(min(odl(space, 1, 2),
                      odl(space, 1, 3), odl(space, 2, 3)))

  # space sortowane po x
  space_x <- space[order(space[,1]),]
  mediana <- floor(r/2)
  # wartosc dla mojej mediany
  xmid <- space_x[mediana,1]

  lewy <- space_x[1:mediana,]
  prawy <- space_x[(mediana+1):r,]

  dL <- Recall(lewy)
  dB <- Recall(prawy)
  dMin <- min(dL, dB)

  # przypadek lewy — prawy podzbior:
  #Band <- space[abs(space[,1] - xmid)<=dMin,]
  Band <- subset(space, abs(space[,1] - xmid)<=dMin)
  m <- nrow(Band)
  if(m<2) return(dMin) # nie ma co sprawdzac

  Band <- Band[order(Band[,2]),] # Band sortowane po y

  for(i in 1:(m-1)){
    for(j in (i+1):m) {
      if(abs(Band[i,2] - Band[j,2]) <= dMin) {
        newd <- odl(Band, i, j)
        if(newd<=dMin) dMin <- newd
      }
    }
  }
}
```

65/71
findClosestPoints <- function(space){
  stopifnot(is.matrix(space), ncol(space)==2,
            is.numeric(space), is.finite(space),
            nrow(space)>0)
  nrow = nrow(space)
  if (nrow<=3) {
    if (nrow==1) dMin = Inf
    if (nrow == 2) {
      dMin = sqrt((space[1,1] - space[2,1])^2 +
                   (space[1,2] - space[2,2])^2)
    }
    if (nrow==3) {
      odl1 = sqrt((space[1,1] - space[2,1])^2 +
                   (space[1,2] - space[2,2])^2)
      odl2 = sqrt((space[1,1] - space[3,1])^2 +
                   (space[1,2] - space[3,2])^2)
      odl3 = sqrt((space[2,1] - space[3,1])^2 +
                   (space[2,2] - space[3,2])^2)
      dMin = min(odl1, odl2, odl3)
    }
  }
  return(dMin) }

xmid <- sort[podzial,1] #wyliczam mediane

# dziel na dwa podzbiorzy
lewy <- sort[1:podzial,]
prawy <- sort[(podzial + 1) : nrow,]
dMin <- min(Recall(lewy), Recall(prawy))

# sprawdzam trzeci przypadek
Band <- sort[abs(sort[,1] - xmid)<=dMin,]
m <- nrow(Band)
if (is.null(m)) m = 0
if (m<2) return(dMin)

# sortuje względem wspolrzednych y-owych
Band <- Band[order(Band[,2]),]

for (i in 1:(m−1)){
  for (j in (i+1):m)){
    if (Band[i,2]−Band[j,2]<=dMin) {
      odleglosc <- sqrt((Band[i,1] − Band[j,1])^2 +
                         (Band[i,2] − Band[j,2])^2)
      if (odleglosc <= dMin) dMin = odleglosc
    }
  }
}
else break
}

dMin
• On our website people submit and evaluate new data,
• Our statistical system should use these pieces of information to give more precise answers,
• Online learning seems to be the solution.
Cost-sensitive learning

- One user is patient and has time to check every single pair returned by our system, even incorrect ones, but certainly does not want to omit even one single plagiarism,
- Second user demands different behavior: only correct pairs should be displayed, even at the cost of some of similar pairs will be omitted,
- cost-sensitive learning seems to be an answer to such demands. It is a machine learning technique which takes into account that some types of misclassifications may be worse than others.
Metric trees

- For now, user can submit a set of functions and get similarities only between them,
- it would be an interesting scenario to find all similar functions, that are in a database, to a submitted one $f_s$,
- usage of all methods is not acceptable for performance reasons,
- so tokens or call count method is used at first,
- but calculating all similarity between $f_s$ and all functions in a database seems still to be not an optimal solution,
- so metric tree is used, which allows to find neighboring functions in $O(\log n)$ time, where $n$ is a number of functions in a database.
How metric tree works? (on the example of vp-tree, images from [13])
Images from [13]
Thank you for your attention.

m.bartoszuk@mini.pw.edu.pl


Aiken, A.: MOSS (Measure of software similarity) plagiarism detection system.


David M. Blei, Andrew Y. Ng, and Michael I. Jordan. **Latent dirichlet allocation.**

T.F. Smith and M.S. Waterman. **Identification of common molecular subsequences.**
Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 

**Weisfeiler-lehman graph kernels.**


Nikolaus Hansen.

Anne Auger and Nikolaus Hansen.

**Tutorial cma-es: Evolution strategies and covariance matrix adaptation.**
