



Anti-plagiarism system for R language

Overview, conclusions and ideas

Maciej Bartoszek

16 czerwca 2016

Wydział Matematyki i Nauk Informatycznych 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 [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 inputted, 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.

FACT

Facts and myths about antiplagiarism systems – Myth #3

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?



Facts and myths about antiplagiarism systems – Fact #2

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

FACT

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)

FACT

SimilaR.Rexamine.com

SimilaR About Antiplagiarism system 2 Admin menu [redacted] logout

Exemplary homework

Submission calculation time: 0h0m4s
Number of pairs: 4

2/4

Hover over pair of functions to see details.

▶ student1.R almostPi (61%)	student2.R almostPi (89%)	Definitely similar ▼
▶ student1.R pythagoreanTriples (44%)	student2.R pythagoreanTriples (40%)	Similar ▼
<pre> student1.R: pythagoreanTriples <- function(m, n) { stopifnot(length(m) == length(n)) a <- m^2 - n^2 b <- 2 * m * n c <- m^2 + n^2 mat <- matrix(c(a, b, c), 3, length(a), byrow = TRUE) l <- split(mat, col(mat)) l[[length(a) + 1]] <- (a^2 + b^2 == c^2 & a * b * c != 0 & a * c > 0) return(l) } student2.R: pythagoreanTriples <- function(m, n) { stopifnot(is.vector(m), is.vector(n), is.numeric(c(m, n))) length(n) == length(m), length(n) > 0, all(c(n - floo m - floor(m)) <= 1e-10), all(c(m, n) >= 0)) a <- m^2 - n^2 b <- 2 * m * n cc <- m^2 + n^2 l <- mapply(c, a, b, cc, SIMPLIFY = FALSE) l[[length(l) + 1]] <- list(a^2 + b^2 == cc^2) l } </pre>		
▶ student1.R almostPi (19%)	student2.R pythagoreanTriples (0%)	✓ Totally different ▼
▶ student1.R pythagoreanTriples (0%)	student2.R almostPi (7%)	✓ Totally different ▼

2/4

Anti-plagiarism - general information

Assumptions

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.

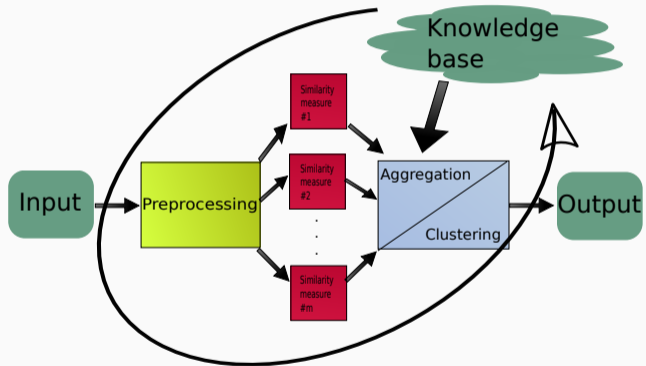


Figure 1: Overview

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.:

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

and

```
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.:

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

and

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

or even

```
1 y <- sqrt(x)
```

Method μ 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 ← 0
2 for(i in x){s ← s + i}
```

and f_2 :

```
1 s ← 0
2 for(i in x){s ← s + i}
3 m ← 0
4 for(i in x){m ← m*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 :

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

and f_2 :

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

and f_3 :

```
1 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 μ_I :

- 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

Tokens

Based on [2, 3].

Let's begin with an example. For function f :

```
1 f <- function(x)
2   {
3     stopifnot(is.numeric(x))
4     y <- sum(x)
5     y
6   }
```

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 `nrow()` (number of rows), `ncol()` (number of columns) and `dim()` (number of rows and columns together),
- Plagiarist can create aliases for functions, e.g. `l <- lapply` and use `l` instead of `lapply`

Possible sequences

- Letters
- Tokens
- Function calls counts
- Program Dependence Graph

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?

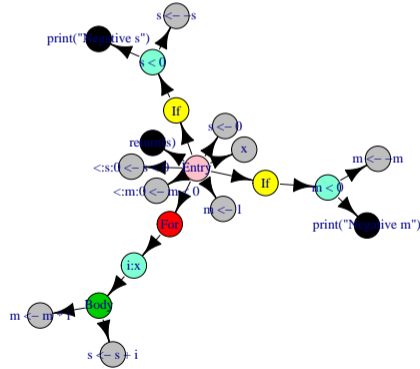
Program dependency graph consists of:

- Control dependency graph
- Data dependency graph

Control dependency graph

Control dependency edges are black.

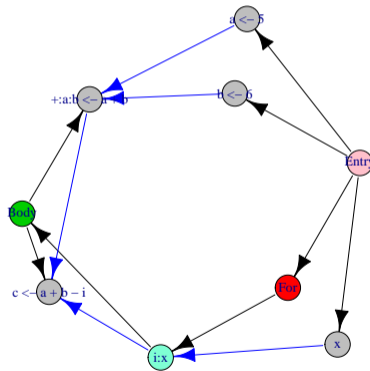
```
1 sum <- function(x)
2 {
3   s <- 0
4   m <- 1
5   for(i in x)
6   {
7     s <- s + i
8     m <- m*i
9   }
10
11  if(s < 0)
12  {
13    s <- -s
14    print("Negative s")
15  }
16  if(m < 0)
17  {
18    m <- -m
19    print("Negative m")
20  }
21  return(s)
22 }
```



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

- 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?

Latent Dirichlet Allocation

LDA [8] fulfills all (or most) requirements.

- „successor” of Latent Semantic Analysis (LSA) method, which is also called Latent Semantic Indexing (LSI)

Latent Dirichlet Allocation

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,

Latent Dirichlet Allocation

LDA [8] fulfills all (or most) requirements.

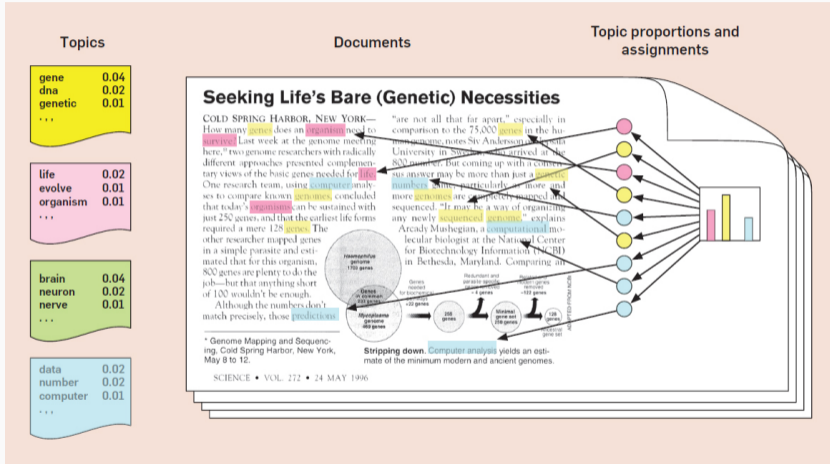
- „successor” of Latent Semantic Analysis (LSA) method, which is also called Latent Semantic Indexing (LSI)
- In the literature every word of source code is used,
- In this work functions names and tokens are used

Latent Dirichlet Allocation

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



Latent Dirichlet Allocation – technical details

- $\alpha = 0.05$ – when α is large, nearly every document will be composed of every topic in significant amounts. In contrast when α is small, each document will be composed of only a few topics in significant amounts,

Latent Dirichlet Allocation – technical details

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

Latent Dirichlet Allocation – technical details

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

Latent Dirichlet Allocation – technical details

- $\alpha = 0.05$ – when α is large, nearly every document will be composed of every topic in significant amounts. In contrast when α is small, each document will be composed of only a few topics in significant amounts,
- β – estimated by the algorithm, larger values of β favor a greater number of words per topic, while smaller values of β favor fewer words per topic.
- Gibbs sampling
- $k = \max\left(\frac{\text{NumberOfFunctions}}{10}, 2\right)$ – topic count

Latent Dirichlet Allocation – technical details

- θ – 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 θ ,
 - 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

$$H = \begin{pmatrix} - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 1 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 1 & 3 & 2 & 3 & 2 & 3 & 2 & 2 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 4 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 6 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 8 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 10 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

Smith-Waterman Algorithm

$$T = \begin{pmatrix}
 & - & A & C & A & C & A & C & T & A \\
 - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 A & 0 & \swarrow & \leftarrow & \swarrow & \leftarrow & \swarrow & \leftarrow & \leftarrow & \swarrow \\
 G & 0 & \uparrow & \swarrow & \uparrow & \swarrow & \uparrow & \swarrow & \swarrow & \uparrow \\
 C & 0 & \uparrow & \swarrow & \leftarrow & \swarrow & \leftarrow & \swarrow & \leftarrow & \leftarrow \\
 A & 0 & \swarrow & \uparrow & \swarrow & \leftarrow & \swarrow & \leftarrow & \leftarrow & \swarrow \\
 C & 0 & \uparrow & \swarrow & \uparrow & \swarrow & \leftarrow & \swarrow & \leftarrow & \leftarrow \\
 A & 0 & \swarrow & \uparrow & \swarrow & \uparrow & \swarrow & \leftarrow & \leftarrow & \swarrow \\
 C & 0 & \uparrow & \swarrow & \uparrow & \swarrow & \uparrow & \swarrow & \leftarrow & \leftarrow \\
 A & 0 & \swarrow & \uparrow & \swarrow & \uparrow & \swarrow & \uparrow & \swarrow & \swarrow
 \end{pmatrix}$$

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),

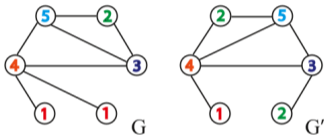
Comparison algorithms

- 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

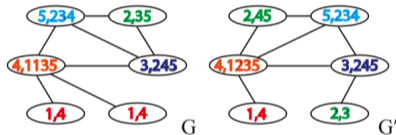
Described in [10].

Given labeled graphs G and G'



1st iteration

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

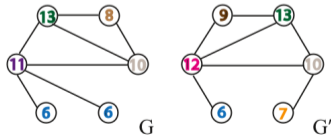


1st iteration
Result of step 3: label compression



1st iteration

Result of step 4: relabeling



End of the 1st iteration

Feature vector representations of G and G'

$$\varphi_{WLsubtree}^{(1)}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$\varphi_{WLsubtree}^{(1)}(G') = (1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1)$$

Counts of
original
node labels

Counts of
compressed
node labels

$$k_{WLsubtree}^{(1)}(G, G') = \langle \varphi_{WLsubtree}^{(1)}(G), \varphi_{WLsubtree}^{(1)}(G') \rangle = 11.$$

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)

- 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 $(FP+FN)/(TP+FP+FN+TN)$,
- Precision is $TP/(TP+FP)$,
- Recall is $TP/(TP+FN)$,
- Accuracy is $(TP+TN)/(TP+FP+FN+TN)$,

Aggregation operators

Our approach is to consider data from 4 methods as a following data frame:

f1 ,	f2 ,	method1 ,	method2 ,	method3 ,	method4 ,	plagiarism
vectorSum ,	listSummation ,	0.6 ,	0.7 ,	0.9 ,	1.0 ,	1
vectorSum ,	regexpTagger ,	0.2 ,	0.4 ,	0.1 ,	0.2 ,	0
.
.
.

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

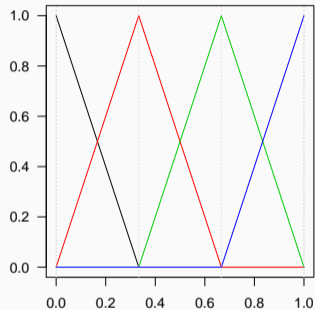
Features		Acc.	Prec.	Rec.	F-meas.	Comment
1	Assym.	0.9893	0.814	0.704	0.755	program dependence graph
	Sym.	0.9881	0.887	0.557	0.684	
7	Assym.	0.9928	0.831	0.863	0.847	tokens – GLCS
	Sym.	0.9925	0.849	0.825	0.837	
1, ..., 19	Assym.	0.9971	0.951	0.931	0.941	all features
	Sym.	0.9967	0.933	0.928	0.931	
1, 2, 7, 9	Assym.	0.9963	0.922	0.924	0.923	standard approach
	Sym.	0.9957	0.912	0.900	0.906	
1, 3, 4, 7, 8, 9, 10	Assym.	0.9977	0.967	0.928	0.947	statistical methods
12, 15, 16, 18, 19	Sym.	0.9971	0.955	0.923	0.939	
1, 2, 3, 4, 6, 7, 9,	Assym.	0.9977	0.969	0.933	0.951	common sense
13, 15, 16, 18, 19	Sym.	0.9975	0.954	0.943	0.949	

Dataset

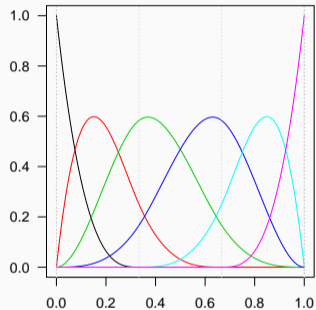
- Dataset from the Example,
- the number of unique observations equals to $m = 30628$,
- the benchmark data set is of the following form:

j	1	2	3	4	5	6	7	8	...
$x_1^{(j)}$	0.82	0.58	0.15	0.37	0.17	0.22	0.69	0.87	...
$x_2^{(j)}$	0.73	0.41	0.25	0.26	0.02	0.13	0.90	0.70	...
$x_3^{(j)}$	0.63	0.84	0.38	0.40	0.11	0.46	0.72	0.83	...
$x_4^{(j)}$	0.92	0.75	0.48	0.39	0.12	0.28	0.80	0.92	...
$y^{(j)}$	1.00	0.75	0.50	0.25	0.00	0.25	0.75	1.00	...

Exemplary spline bases



(a) $p = 1, k = 2$

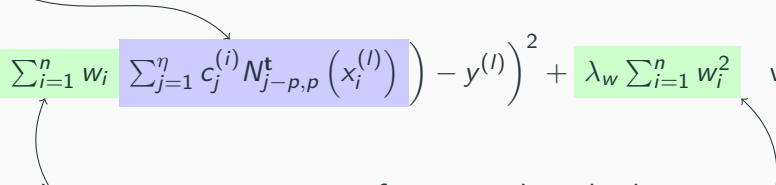


(b) $p = 3, k = 2$

Figure 2: Exemplary B-spline basis functions $N_{j-p,p}^{\mathbf{t}}(x)$ as a function of x , $j = 1, \dots, p + k + 1$; \mathbf{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 \mathbf{w} , optimizes for \mathbf{c} and in fact can be written in the form of a standard quadratic programming task with linear constraints

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


- the outer-level component, optimizing for \mathbf{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 \mathbf{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 δ_2^2 (b).

method	accuracy	precision	recall	F	δ_2^2
Proposed method (a)	0.997	0.921	0.933	0.927	106.62
Proposed method (b)	0.997	0.900	0.920	0.910	95.85
Linear regression	0.995	0.810	0.969	0.883	103.53
Logistic regression	0.997	0.885	0.960	0.921	—
Random forest	0.998	0.927	0.956	0.941	—

Performance

Table 2: Performance measures as functions of different weighting vectors; $p = 3$, $k = 1$, $\lambda_w = 0$, with and without idempotization.

w_1	w_2	w_3	w_4	accuracy	precision	recall	F	∂_2^2	idempot.
1	0	0	0	0.992	0.848	0.693	0.763	186.30	Yes
0	1	0	0	0.995	0.927	0.787	0.851	208.74	Yes
0	0	1	0	0.994	0.803	0.853	0.828	316.04	Yes
0	0	0	1	0.996	0.904	0.840	0.871	136.67	Yes
0.27	0.06	0.38	0.29	0.996	0.952	0.800	0.870	137.34	No
0.41	0.12	0.07	0.40	0.997	0.919	0.907	0.913	107.69	Yes

Performance

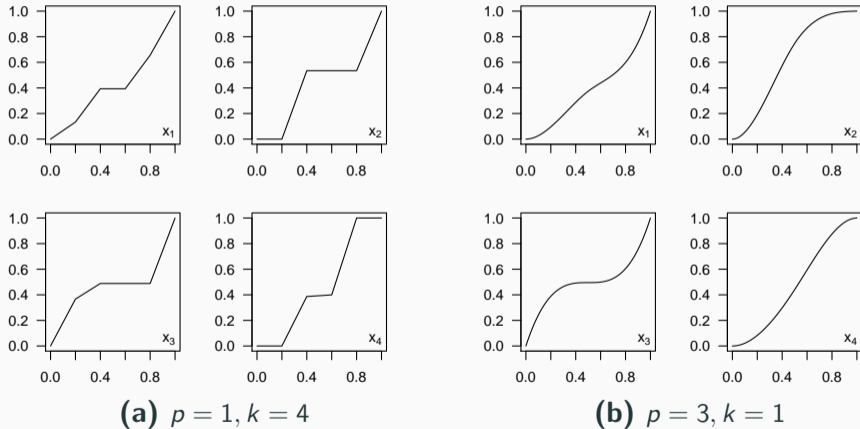
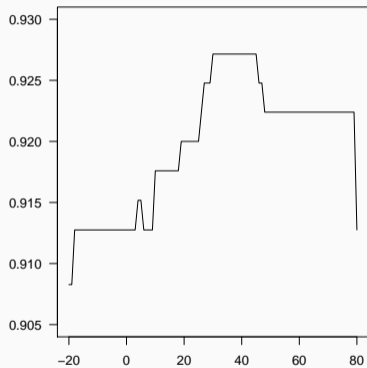
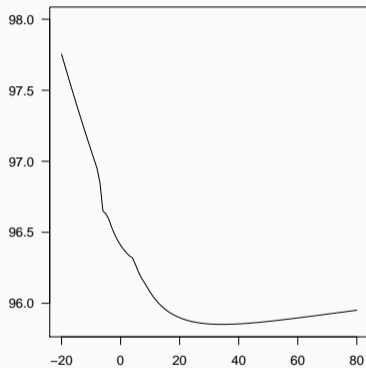


Figure 3: Best B-splines of different degrees fit to the training sample.

Performance



(a) F -measure (greater is better)



(b) σ_2^2 (less is better)

Figure 4: F -measure and squared error as a function of the λ_w regularization coefficient.

Presentation of antiplagiarism results to an end user

Clustering

- 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:

```
1 odl<-function(space,x1,x2){
2   sqrt((space[x1,1]-space[x2,1])^2+
3     (space[x1,2]-space[x2,2])^2)}
4
5 findClosestPoints <- function(space){
6   stopifnot(is.matrix(space), is.numeric(space),
7     ncol(space)==2,nrow(space)>0,all(is.finite(space)))
8
9   r <- nrow(space)
10  if(r==1) return(Inf)
11  if(r==2) return(odl(space,1,2))
12  if(r==3) return(min(odl(space,1,2),
13    odl(space,1,3), odl(space,2,3)))
14
15  #space sortowane po x
16  space_x <- space[order(space[,1]),]
17
18  mediana <- floor(r/2)
19  #wartosc dla mojej mediany
20  xmid <- space_x[mediana,1]
21
22  lewy <- space_x[1:mediana,]
23
24  prawy <- space_x[(mediana+1):r,]
25
26  dL <- Recall(lewy)
27  dR <- Recall(prawy)
28  dMin <- min(dL,dR)
29
30  #przypadek lewy - prawy podzbiór:
31  #Band <- space[abs(space[,1]-xmid)<=dMin,]
32  Band <- subset(space, abs(space[,1]-xmid)<=dMin)
33  m <- nrow(Band)
34
35  if(m<2) return(dMin) #nie ma co sprawdzac
36
37  Band <- Band[order(Band[,2]),] #Band sortowane po y
38
39  for(i in 1:(m-1)){
40    for(j in (i+1):m) {
41      if(abs(Band[i,2]-Band[j,2]) <= dMin) {
42        newd <- odl(Band,i,j)
43        if(newd<=dMin) dMin <- newd }}}
44  }
45 }
```


Example

Aleksandra B [REDACTED]:

```
1 findClosestPoints <- function(space){
2
3   stopifnot(is.matrix(space), ncol(space)==2,
4             is.numeric(space), is.finite(space),
5             nrow(space)>0)
6   nrow = nrow(space)
7   if (nrow<=3) {
8     if (nrow==1) dMin = Inf
9     if (nrow == 2) {
10      dMin = sqrt((space[1,1] - space[2,1])^2+
11                (space[1,2] - space[2,2])^2) }
12    if (nrow == 3) {
13      odl1 = sqrt((space[1,1] - space[2,1])^2+
14                 (space[1,2] - space[2,2])^2)
15      odl2 = sqrt((space[1,1] - space[3,1])^2+
16                 (space[1,2] - space[3,2])^2)
17      odl3 = sqrt((space[2,1] - space[3,1])^2+
18                 (space[2,2] - space[3,2])^2)
19      dMin = min(odl1, odl2, odl3)}
20    return(dMin) }
21  # sortuje wzgledem wspolrzecznych x-owych
22  sort <- space[order(space[,1]),]
23  podzial <- floor(nrow/2)
```

```
23   xmid <- sort[podzial,1] #wyliczam mediane
24
25   #dziele na dwa podzbiory
26   lewy <- sort[1:podzial, ]
27   prawy <- sort[(podzial + 1) : nrow, ]
28   dMin <- min(Recall(lewy), Recall(prawy))
29   # sprawdzam trzeci przypadek
30   Band <- sort[abs(sort[,1]-xmid)<=dMin,]
31   m <- nrow(Band)
32   if (is.null(m)) m = 0
33   if (m<2) return(dMin)
34   # sortuje wzgledem wspolrzecznych y-owych
35   Band <- Band[order(Band[,2]),]
36
37   for (i in 1:(m-1)){
38     for (j in ((i+1):m)){
39       if (Band[i,2]-Band[j,2]<=dMin) {
40         odlegosc <- sqrt((Band[i,1] - Band[j,1])^2+
41                          (Band[i,2] - Band[j,2])^2)
42         if (odlegosc <= dMin) dMin = odlegosc
43       }
44     }
45   }
46   dMin
```

Statistical system

- 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.

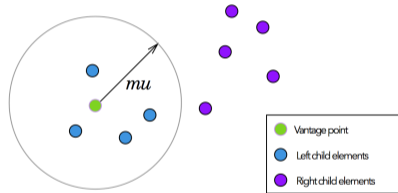
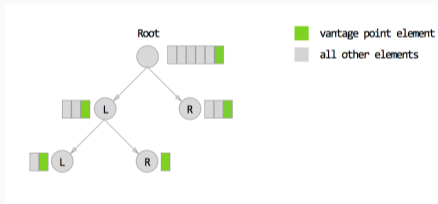
Cost-sensitive learning

- 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 a demands. It is a machine learning technique which takes into account that some types of misclassifications may be worse than others.

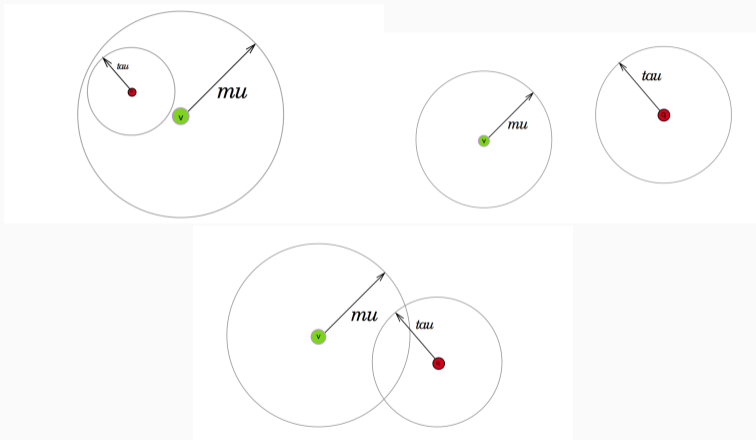
Metric trees

- 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

References I




-  R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2014).
<http://www.R-project.org/>.
-  Prechelt, L., Malpohl, G., Philippsen, M.: JPlag: Finding plagiarisms among a set of programs. Tech. rep. (2000).
-  Prechelt, L., Malpohl, G., Philippsen, M.: Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science 8(11), 1016–1038 (2002).

References II


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

<http://theory.stanford.edu/~aiken/moss/>.
-  Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Trans. Program Lang. Syst. 9(3), 319–349 (1987).
-  Liu, Ch., Chen, C., Han, J., Yu, P.S.: GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis. In: Proc. 12th ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining (KDD'06), 872–881 (2006).

References III

-  Qu, W., Jia, Y., Jiang, M.: Pattern mining of cloned codes in software systems. *Information Sciences* 259 (2014), 544–554.
-  David M. Blei, Andrew Y. Ng, and Michael I. Jordan.
Latent dirichlet allocation.
J. Mach. Learn. Res., 3:993–1022, March 2003.
-  T.F. Smith and M.S. Waterman.
Identification of common molecular subsequences.
Journal of Molecular Biology, 147(1):195 – 197, 1981.

References IV

 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt.

Weisfeiler-lehman graph kernels.

J. Mach. Learn. Res., 12:2539–2561, November 2011.

 Nikolaus Hansen.

***The CMA Evolution Strategy: A Comparing Review*, pages 75–102.**

Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

References V

-  Anne Auger and Nikolaus Hansen.
Tutorial cma-es: Evolution strategies and covariance matrix adaptation.
In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '12*, pages 827–848, New York, NY, USA, 2012. ACM.
-  <http://www.huynh.com/posts/similarity-search-101-with-vantage-point-trees/>.