



Be Wise, Plagiarize

Standard similarity detection

Karma Tarap, Programmer | Budapest, Oct 2012

Disclaimer



The opinions expressed in this presentation and on the following slides are solely those of the presenter and not necessarily those of Novartis. Novartis does not guarantee the accuracy or reliability of the information provided herein



“Plagiarism detection is the process of locating instances of plagiarism within a work or document.”

– Wikipedia

■ Plagiarism detection algorithms:

1. Well researched area. Used in:
 - Academia to identify cheating
 - Industry to identify copyright infringements
2. Has the goal: “How similar are a set of documents”



- Standard programs are an essential component of clinical trial reporting.
 1. Are the standards being used?
 2. What is the degree of modifications required?



On a fundamental level, we are interested in finding:

“How similar are a set of documents?”

How can we program this?

Apply plagiarism detection techniques to our standard similarity problem.

The main difference being: In “plagiarism detection” a high score = bad. Whereas, in our case a high score = good.

Some considerations



Lets consider the following 3 code snippets:

<pre>proc sort data=class; by age; run;</pre> 1	<pre>data class.proc ; sort = ' by age ' ; run ;</pre> 2	<pre>/*proc sort data=class; by age run;*/</pre> 3
--	---	---

A word by word comparison would yield a high match for all of the above, despite being functionally different.

Some considerations (purpose)



Lets consider the following 3 code snippets:

<pre>proc sort data=class; by age; run;</pre> 1	<pre>data class.proc ; sort = ' by age ' ; run ;</pre> 2	<pre>/*proc sort data=class; by age ;run;*/</pre> 3
--	---	--

A word by word comparison would yield a high match for all of the above, despite being functionally different.

Purpose matters

Some considerations (context)



Lets consider the following 3 code snippets:

<pre>proc sort data=class; by age; run;</pre> 1	<pre>data class.proc ; sort = ' by age ' ; run ;</pre> 2	<pre>/*proc sort data=class; by age ;run;*/</pre> 3
--	---	--

A word by word comparison does not take into consideration, special meaning generated by context.

Context matters

Some considerations (order)



Lets consider the following 3 code snippets:

<pre>proc sort data=class; by age; run;</pre> 1	<pre>;proc sort data=class; by age; run;</pre> 2	<pre>/*proc sort data=class; by age run;*/</pre> 3
--	---	---

Comparing files based on the index of the word yields a complete mismatch of the above programs

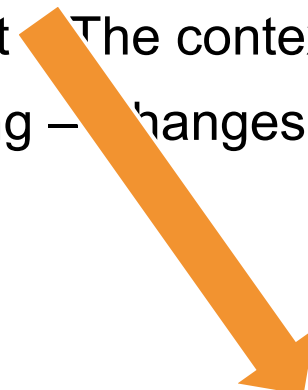
Order doesn't matter

Some considerations (cont.)



- The issues identified in this approach can be classified as follows:

1. Purpose – The purpose of the word
2. Context – The context of the word given the surrounding words
3. Ordering – Changes of order of sections in a file



Tokenization



- Tokens are the basic elements of a language
- SAS defines four basic token types:
 1. **Literal** - One or more characters enclosed in single or double quotation marks.
 2. **Name** - One or more characters beginning with a letter or an underscore.
 3. **Number** - A numeric value.
 4. **Special character** - Any character that is not a letter, number, or underscore

We will need to extend this a little further (keywords , macro...)

Tokenization flow

Tokenization is the process of breaking a language into tokens.

```
%let pgmname = aeder;  
data aev0 ;  
  set data_a.a_aev ;  
  *For unscheduled visits, use the repeat visit number ;  
  vis_ln = ifN(visln eq 999, rpevisln, visln) ;  
  *Compress out hidden characters from AE names;  
  aevnamla = compress(aevnamla, 'kw') ;  
run;
```

code

[mKeyword, bIdentifier, Operator, bIdentifier, bKeyword,
bIdentifier, bKeyword, bIdentifier, Punctuation,
bIdentifier, bIdentifier, Operator, bIdentifier,
Separator, bIdentifier, bIdentifier, Number, Separator,
bIdentifier, Separator, bIdentifier, Separator,
bIdentifier, Operator, bIdentifier, Separator, Identifier,
Separator, Operator, Literal, Separator, Keyword]

tokens

1- Literal	4- bIdentifier	7- Operator
2- Number	5- mKeyword	8- Separator
3- bKeyword	6- mIdentifier	9- Punctuation

mapping

[5, 4, 7, 4, 3, 4, 3, 4, 9, 4, 4, 7, 4, 8, 4, 4, 2, 8, 4, 8, 4, 8, 4, 7, 4, 8, 4, 8, 8, 1, 8, 3]

Abbreviated tokens

Now resistant to datastep and variable name changes!

Some considerations (cont.)



- The issues identified in this approach can be classified as follows:
 1. Purpose – The purpose of the word
 2. Context – The context of the word given the surrounding words
 3. Ordering – Changes of order of sections in a file



n-grams

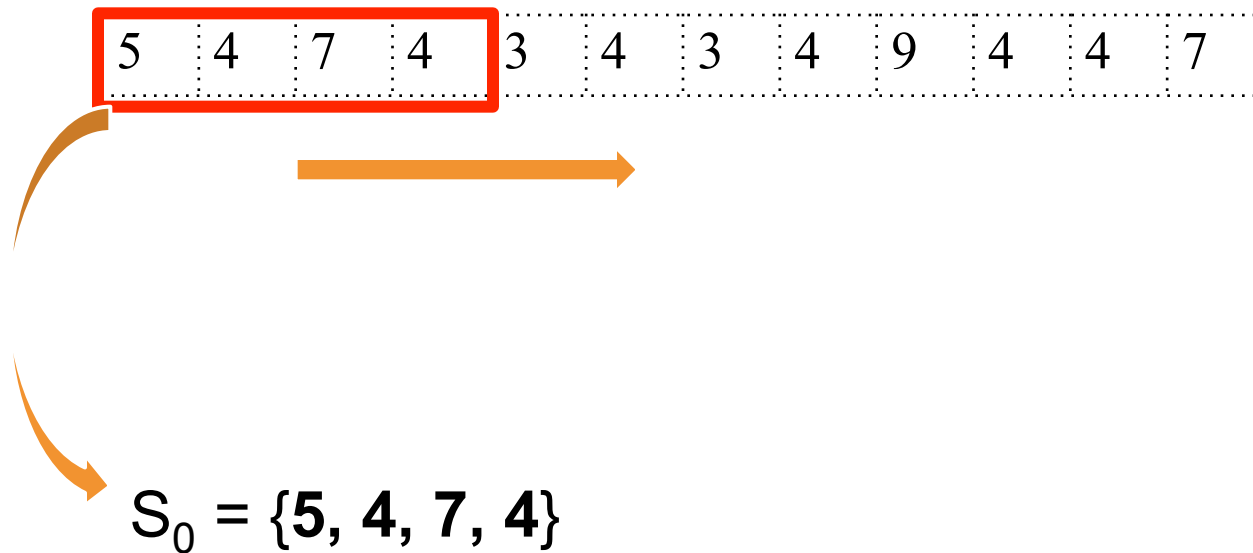


- An n-gram is a contiguous sequence of n items from a given sequence of text
- Converting our tokens to n-grams allows us to compare **sections** of code.

n-grams sliding window



Let $n = 4$



n-grams sliding window



Let $n = 4$

5	4	7	4	3	4	3	4	9	4	4	7
---	---	---	---	---	---	---	---	---	---	---	---

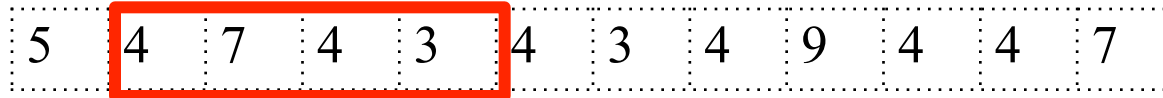

$$S_0 = \{5, 4, 7, 4\}$$


$$S_1 = \{4, 7, 4, 3\}$$

n-grams sliding window



Let $n = 4$



We can now compare
n-grams of files
instead of single
tokens.

$$S_0 = \{5, 4, 7, 4\}$$

$$S_1 = \{4, 7, 4, 3\}$$

$$S_2 = \{7, 4, 3, 4\}$$

$$S_n = \{\dots\dots\}$$

Some considerations (cont.)



- The issues identified in this approach can be classified as follows:
 1. Purpose – The purpose of the word
 2. Context -- The context of the word given the surrounding words
 3. Ordering – Changes of order of sections in a file



Jaccard's
Index

We will also now
look at scoring.

Jaccard's Index



- Jaccard's Index is a statistic for comparing the similarity of sets.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

- Intersect of files A and B, divide by their union.
- Has a bound of 0 to 1.
- By comparing n-grams irrespective of their position, we have an order independent comparison.

Jaccard's Index (cont.)



An example:

- File A: {5, 4, 7, 4} {3, 4, 3, 4} {9, 4, 4, 7}
- File B: {5, 4, 7, 4} {3, 4, 3, 4} {3, 4, 5, 7}

$A \cup B$ = Total distinct n-grams=4, $A \cap B$ = total matched n-grams=2

- *$J(A,B)=2/4 =.5$*

Similarity between file A and file B is 50%



- Apply plagiarism detection techniques to our standard similarity problem
 1. Purpose – Tokenization
 2. Context – n-grams
 3. Ordering – Jaccard's Index
- Implement solution in Proc Groovy (SAS 9.3)
 - Full code provided in the paper appendix

Results



Similarity Report

Overall Summary

Threshold value: 95.00%

Sets sensitivity of match

File comparison Summary

Project	# Above threshold	# Total files in project	% Similarity
M K2201	52	55	94.55%
M K2101	33	38	86.84%
M K2102	33	39	84.62%
M K2103	34	40	85.0%

High level summary checks if standards are being used

Extended Report

File comparison breakdown

Project	File being compared	% Max Similarity	Closest matching file(s)
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/a_cycle.sas	0.45	/vob/CE J398X/CE J398X2101/report/pgm_a/stl1_trt.sas
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/a_ecg.sas	0.61	/vob/CE J398X/CE J398X2101/report/pgm_a/stl1_trt.sas
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/a_dar.sas	0.85	/vob/CE J398X/CE J398X2101/report/pgm_a/a_vsnab1.sas
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/a_dar.sas	0.88	/vob/CE J398X/CE J398X2101/report/pgm_a/a_vsnab1.sas
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/ha_a_liver_les.sas	0.91	/vob/CE J398X/CE J398X2101/report/pgm_a/autorun.sas
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/ha_a_liver_les.sas	0.91	/vob/CE J398X/CE J398X2101/report/pgm_a/trt.sas
M K2101	/vob/CM K162X/CM K162X2101/report/pgm_a/ha_a_liver_les.sas	0.91	/vob/CE J398X/CE J398X2101/report/pgm_a/a_ident.sas

Low level breakdown identifies standards that require updating.



1. Are the standards being used?
 - Is the user aware they exist?
 - Is the outputs/datasets required not standard?
 - Is the standard not flexible enough?

2. What is the degree of modifications required?
 - Few modifications suggest the standard programs are robust
 - Many changes suggest the programs need updating

Questions?



Be wise, plagiarize

Sample Groovy code



```
proc groovy ;
submit ;

class SimpleTokenizer {
    def tokenListAb = []
    //def tokenList = [];

    def baseKeywords = ['libname', 'label', 'infile', 'filename', 'set',
        'file', 'by', 'proc', 'run', 'do', 'end', 'while', 'until', 'if',
        'else', 'cards', 'datalines', 'quit', 'data'] //expand to suit need

    def macroKeywords = [%abort, %copy, %do, %until, %while,
        %end, %global, %goto, %if, %then, %else, %input,
        %let, %local, %macro, %mend, %put, %return, %symdel,
        %syscall, %sysexec, %syslput, %sysrput, %window]

    //First break into top level groups - string literal,
    //identifier, specials, numerics and whitespace
    def re = '''(?si)(?:"[^\"]*"|' '[^']*'|%?[a-z_0-9]+\|\\p{Punct}|\\d+|\\s+|.*)'''

    def handleStatement(def statement){
        def scanner = statement =~ re
        scanner.each{match -> handleToken match}}

    def handleToken(def token) {
        //println 'Tokens '+ token
        switch(token) {
            // String literal can span multiple lines
            case ~/(?s)^\s*(?:\.'|\\").*$$/:
                //tokenList << 'Literal'
                tokenListAb << 1
                break
            // Numbers
            case ~/^\\d+$/:
                tokenListAb << 2
                //tokenList << 'Number'
                break
            // Identifiers
            case ~/^\\[A-Za-z_0-9]+$/:
                if (token.toLowerCase() in baseKeywords){
                    //tokenList << 'bkeyword'
                    tokenListAb << 3
                }
            default:
                //tokenList << 'Other'
                tokenListAb << 4
        }
    }
}
```