

ENSURING PROGRAMMING INTEGRITY WITH PYTHON: DYNAMIC CODE PLAGIARISM DETECTION

February 27th, 2019

Copyright © 2018 Covance. All Rights Reserved

Public



COVANCE
SOLUTIONS MADE REAL®

Michael Stackhouse

BIOGRAPHY

Michael Stackhouse holds a bachelor's degree from Arcadia University where he studied Business Administration, Economics, and Statistics. He is currently an MIDS student at UC Berkeley School of Information, studying Data Science.

Mr. Stackhouse has extensive CDISC experience, working with both SDTM and ADaM standards as well as serving as a subject matter expert for Define.xml. Mr. Stackhouse's favorite career projects include macro and utility development, as well as molding the minds of young programmer Padawans into the programming Jedis of the future.

Mr. Stackhouse is currently a Manager of Statistical Programming at Covance and father to three young, furry, four-legged children.

About Covance

Covance Inc., the drug development business of Laboratory Corporation of America Holdings (LabCorp), is the world's most comprehensive drug development company.

We are dedicated to advancing healthcare and delivering Solutions Made Real® by providing high quality nonclinical, clinical, commercialization and informatics services to pharmaceutical and biotechnology companies to help increase the speed, precision and effectiveness of drug development.

Covance is currently working on about half of all active clinical trials globally and has been involved in all top 50 best-selling drugs on the market.

Introduction

I have two dogs.

I have one cat.

=?

Introduction

- ▶ Programmatically simple to check equality

```
In [1]: "I have two dogs." == "I have one cat."  
Out[1]: False
```

Introduction

DOCUMENT1.TXT

```
I have two dogs.  
I have one cat.  
My dog's names are Dash and Kara.  
My cat's name is Reggie.
```

DOCUMENT2.TXT

```
I have two dogs.  
They like to play.  
I have one cat.  
  
My dog's names are Dash and Kara.  
My cat's name is Reggie.
```

Diff Output

```
>>> diff document1.txt document2.txt  
1a2  
> They like to play.  
2a4
```

Introduction

DOCUMENT1.TXT

```
I have two dogs.  
I have one cat.  
My dog's names are Dash and Kara.  
My cat's name is Reggie.
```

DOCUMENT3.TXT

```
I have one cat.  
I have two dogs.  
My cat's name is Reggie.  
My dog's names are Dash and Kara.
```

Diff Output

```
>>> diff document1.txt document3.txt  
1d0  
< I have two dogs.  
3c2  
< My dog's names are Dash and Kara.  
---  
> I have two dogs.  
4a4  
> My dog's names are Dash and Kara.
```

Motivation

Similar – not equal

Motivation

Covance **DETECT**

Dynamic Evaluation of Technical Conformance Transgressions

Agenda

- ▶ Walk through of versions 1 through 4
 - Distance measures of text
 - Multiprocessing
 - Text vectorization
- ▶ Why Python?
 - Powerful string operations
 - Simple file handling
 - Scikit-learn libraries

Version 1

- ▶ Measuring **document** similarity
 - Token Sort Ratio
 - Generally interpret as a percent similarity

```
In [1]: from fuzzywuzzy import fuzz
In [2]: fuzz.token_sort_ratio("I have a dog", "I have a cat")
Out[2]: 75
```

```
In [3]: fuzz.token_sort_ratio("I have a dog", "DOG A HAVE I")
Out[3]: 100
```

Version 1

- ▶ Why does this work with SAS code?
 - `if a=b` essentially means the same thing as `if b=a`
 - SAS syntax is not case sensitive
- ▶ Why doesn't this approach work?
 - SAS syntax is inherently similar
 - It doesn't tell you **where** the programs are similar

Version 2

► Goal

- Find the location of similarities between the two documents

► Method

- Check similarity by line instead of by document
- Brute force – every line in production is compared to every line in validation

Version 2

► Why by line?

- Very easy to read a program by lines in Python

```
with open('program.sas', 'r') as f:  
    text_in_list = f.readlines()
```

- Easily traceable back to the original document
 - Most editors will allow you to go directly to a line in the file

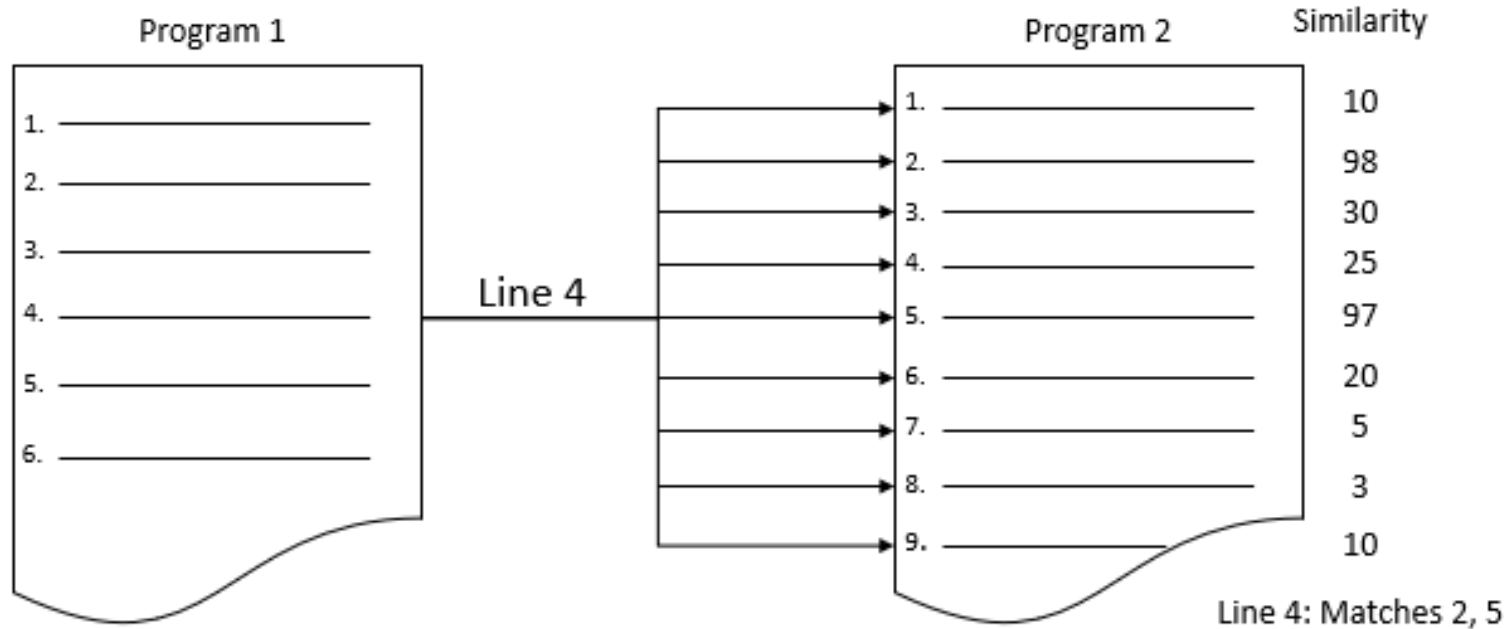
Version 2

Algorithm Pseudocode

```
# Start a dictionary variable
list_of_matches = {}

# Loop over all lines in Program 1
for line_a in Program 1:
    # Loop over all lines in Program 2
    for line_b in Program 2:
        # Check the similarity
        similarity = distance(line_a, line_b)
        # If the lines are similar enough, track the number
        if similarity > threshold:
            list_of_matches[line number of line_a].append(line number of line_b)
```

Version 2



Version 2

Algorithm Pseudocode

```
# Start a dictionary variable
list_of_matches = {}

# Loop over all lines in Program 1
for line_a in Program 1:
    # Loop over all lines in Program
    for line_b in Program
        # Check the similarity
        similarity = distance(line_a, line_b)
        if len(line_a) > 7:
            # If the lines are similar enough, track the number
            if similarity > threshold:
                list_of_matches[line number of line_a].append(line number of line_b)
```

Version 2

Sample Output

```
Similar to: [131]-----> %macro _rename(str, var_out);  
Similar to: [132]----->     %do i=1 %to %sysfunc(countw(&str));  
Similar to: [133]----->     %let item=%scan(&str,&i);  
Similar to: [134]----->     %let item2=%scan(&var_out,&i);  
----->     &item = &item2  
----->     %end;
```

Version 2

► Problems

- Runtime of a long program can be >4 minutes
 - Brute force approach raises algorithm complexity (i.e. 1000 line dev X 1000 line QC = 1,000,000 line comparisons)
- Output is still fairly extensive
 - Information is more useful – but still a good deal of noise

Version 3

► Goal

- Speed up the algorithm
- Give a high level summary to help find problem areas

► Approach

- Capture some summary information about each program and plot a graph
- Parallelize the comparisons

Version 3

► Multiprocessing

- The challenge
 - Global Interpreter Lock (GIL)
 - Locks so that only one thread can hold control of the Python Interpreter at a time
 - Not taking advantage of the full power of the CPU available
 - Constraint is real-time in this case – not CPU or memory
- Workaround
 - Python multiprocessing library

Version 3

```
import multiprocessing as mp

def f(x):
    return x

if __name__ == '__main__':
    with mp.Pool(4) as p:
        pools = [p.apply_async(f, args=(x,)) for x in range(4)]

        for x in pools:
            try:
                results.append(x.get())
            except:
                results = [x.get()]

    print(results)
    print("Done")
```

Result:

```
>>> [0, 1, 2, 3]
>>> Done
```

Version 3

Program 1, Section 1

1.	_____
2.	_____
3.	_____
4.	_____
5.	_____
6.	_____

Program 1, Section 2

1.	_____
2.	_____
3.	_____
4.	_____
5.	_____
6.	_____

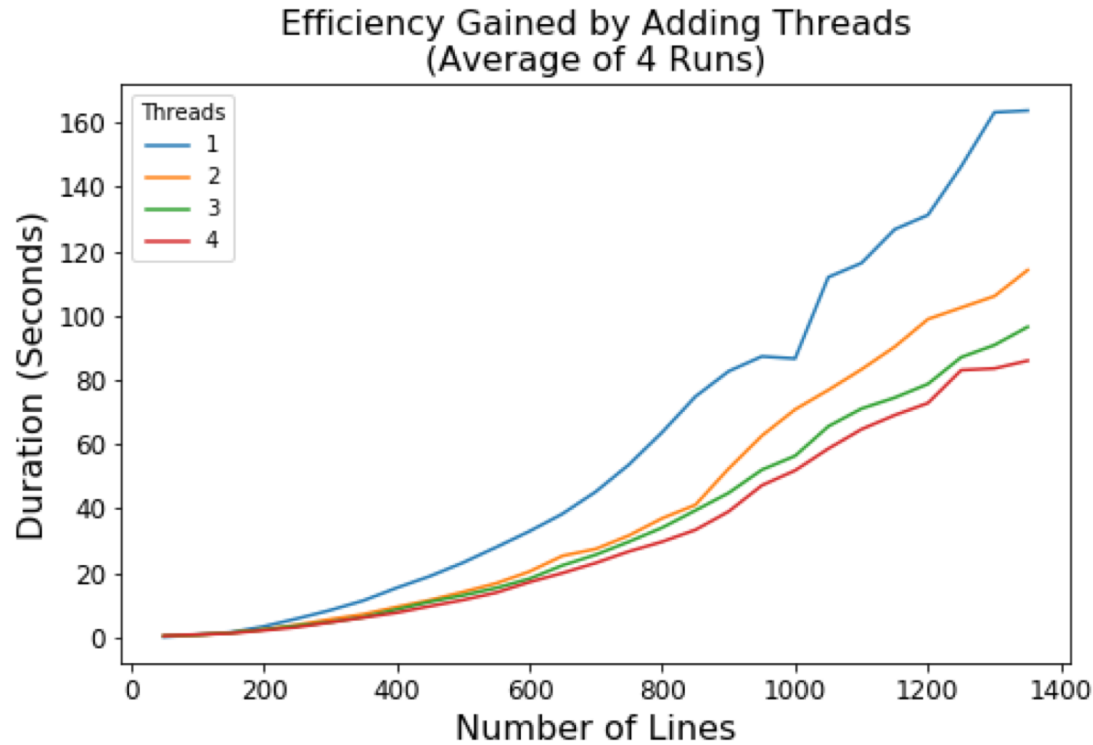
Program 1, Section 3

1.	_____
2.	_____
3.	_____
4.	_____
5.	_____
6.	_____

Program 2

→	1.	_____
→	2.	_____
→	3.	_____
→	4.	_____
→	5.	_____
→	6.	_____
→	7.	_____
→	8.	_____
→	9.	_____

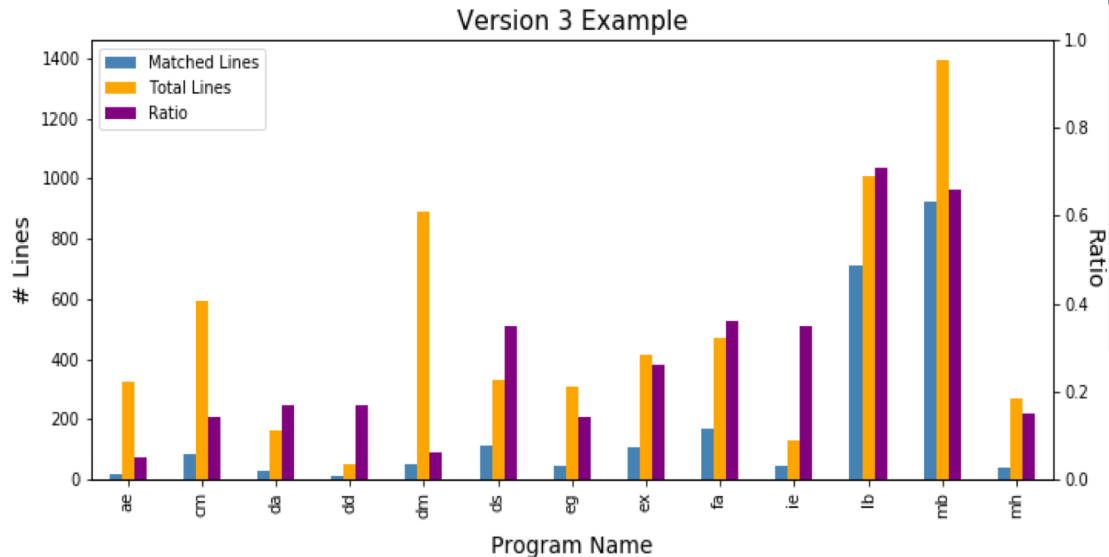
Version 3



Version 3

Summary Chart

- Total Lines in Program 1
- Matched Lines in Program 1
- Ratio of Matched Lines to Total



Version 4

- ▶ Significant improvements, including:
 - A different choice of distance metric
 - Dramatically enhanced speed
 - More detailed metadata per program
 - Improved output delivery format
 - Improved foundation for further development
- ▶ Need to detour to discuss some new concepts

Text Vectorization

- ▶ How do you do quantitative analysis on text?
 - Text vectorization

```
string1 = "I have a dog and his name is Dash"  
string2 = "I have a cat and his name is Reggie"
```

	I	HAVE	A	DOG	CAT	AND	HIS	NAME	IS	DASH	REGGIE
string1	1	1	1	1	0	1	1	1	1	1	0
string2	1	1	1	0	1	1	1	1	1	0	1

Measuring Similarity

- ▶ Instead of the token sort ratio, version 4 uses Cosine Similarity
 - Results are very similar between the two
 - Cosine Similarity is very computationally efficient when working with the text vectors
 - Calculated with a dot product and leverages Python libraries written in lower level languages

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Measuring Similarity

```
# Set the strings to be compared
string1 = "I have a dog and his name is Dash"
string2 = "I have a cat and his name is Reggie"

# Display the results for both cosine similiary and token_sort_ratio
print("Cosine similarity = {}".format(\
    get_cosine_similarity(string1, string2)))
print("Token sort ratio = {}".format(\
    fuzz.token_sort_ratio(string1, string2)))
```

Results:

Cosine similarity = 0.7777777777777779

Token sort ratio = 76

**Note: see paper for full code*

Speed Improvements

- ▶ Speed dramatically improved using this method
 - ~99% increase over multiprocessing speeds
- ▶ Improvements due to a few factors
 - Cosine similarity more computationally efficient
 - Text vectorization is done up front and only once
 - Old method had to tokenize each program for each comparison
 - An 1000 line program would do this step 999 times too many

Other Factors in Redesign

- ▶ Program entirely rewritten in an object oriented design
 - Makes capturing more metadata around each program much easier

Summary Level	Line Level
Total Lines	Original Line
Number of matched lines	Matched lines
Matched line percentage	Matched line similarities
Number of short match filtered lines	Top matches
Short match filtered percentage	Short match filtered lines
Number of blocks	Block Indicator
	Block ID

Blocks

PROGRAM 1

```
data x;  
    set y (where=(var1 ^= var2)  
keep=var1 var2);  
run;
```

```
proc sort data=x;  
    by var1 var2;  
run;
```

```
data z;  
    merge x (in=a) y (in=b);  
    by var1 var2;  
    if x;  
run;
```

PROGRAM 2

```
proc sql;  
    create table x as  
        select var1, var2 from y  
order by var1, var2;  
quit;
```

```
data z;  
    merge x (in=a) y (in=b);  
    by var1 var2;  
    if x;  
run;
```

Improved Output Format

- ▶ Instead of plots and text documents, moved everything into an Excel document
- ▶ The Summary Tab
 - Aggregate level attributes are displayed about each program
- ▶ The Program Match Tabs
 - Contains a full print out of each line in Program 1 paired with its **top match** to Program 2
 - Additional line level attributes are also presented.

Note: Examples of each of these tabs can be seen in Appendix 2 of the paper

Where To Go From Here

- ▶ Leverage human review to create training data to apply machine learning
 - Great deal of nuance where if/then/else gets too complex
 - Machine learning could be trained to classify issues against expected similarities
 - Direct variable matchings will probably be similar
 - Complex derivations, unique variable naming, complex expressions will not be

What this *does not* do

- ▶ DETECT does **NOT** tell you why a match exists
 - Specs may have been written with code in the definitions
 - Code may have been provided in the SAP
 - Code may have taken from completely separate programs developed by the same person, who used the same programming logic (i.e. ISO8601 date formatting)
- ▶ It does **NOT** tell you who copied whom

Conclusion

- ▶ One line of maliciously copied code can wreak havoc on a programming team
- ▶ DETECT helps ensure programming integrity by evaluating independence
- ▶ Future improvements will help sift through noise in the output even more clearly
- ▶ DETECT can find the “what” – you need to find the “why”

Questions?

Michael Stackhouse

Covance, Inc.

4000 Centergreen Way # 300

Cary, NC 27513

Michael.Stackhouse@Covance.com

<https://www.linkedin.com/in/michael-s-stackhouse/>



Covance Inc., headquartered in Princeton, NJ, USA, is the drug development business of Laboratory Corporation of America Holdings (LabCorp). COVANCE is a registered trademark and the marketing name for Covance Inc. and its subsidiaries around the world.