Stylistic fingerprints

- Stylometry has been applied to:
  - Fine-art
  - Music
  - Unconventional text
  - Translated text
  - Source code

Supervised stylometry

- Given a set of documents of known authorship, classify a document of unknown authorship
  - Classifier trained on undisputed text
- Scenario: Alice the Anonymous Blogger vs. Bob the Abusive Employer
  - Alice blogs about abuses in Bob's company
    - Blog posted anonymously (Tor, pseudonym, etc)
  - Bob obtains 5,000 words of each employee's writing
    - Bob uses stylometry to identify Alice as the blogger

theconnor: Cisco just offered me a job! Now I have to weigh the utility of a **fatty** paycheck against the daily commute to San Jose and hating the work.

about 19 hours ago from web · Reply · View Tweet

channel partner advocate for Cisco Alert

Fingerprints in textual data

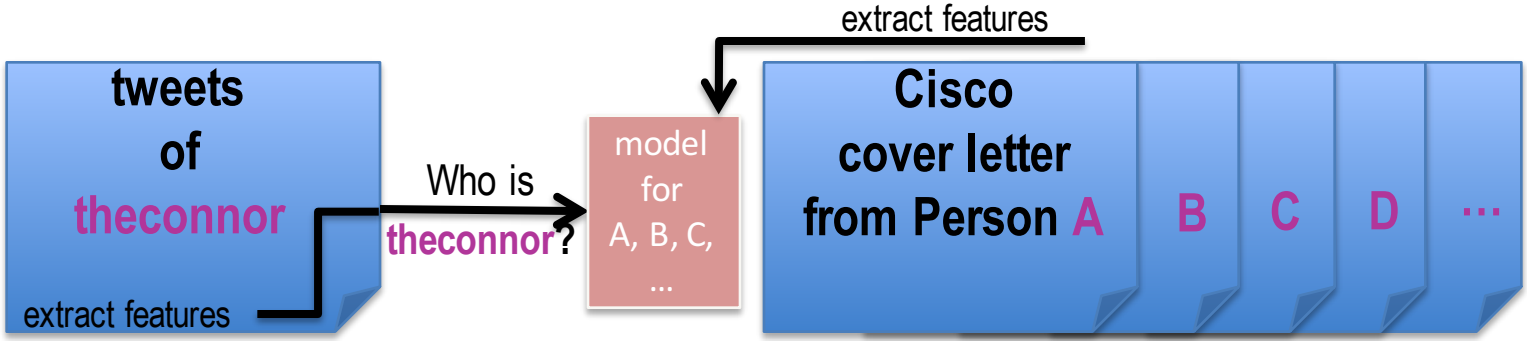
- **Stylistic fingerprints**

theconnor: Cisco just offered me a job! Now I have to weigh the utility of a **fatty** paycheck against the daily commute to San Jose and hating the work.
about 19 hours ago from web · Reply · View Tweet

Frequency of punctuation

Frequency of function words

  - Use stylometric fingerprints to find who "**theconnor**" is.
  - Collect the rest of the tweets in the timeline, compare to the cover letters submitted to Cisco and identify **theconnor**.



theconnor made her Twitter profile private
and deleted all information on her homepage
right after the event
but it was too late
since search engines cache search results which can lead to old information.

Search Twitter

Have an account? Log in

TWEETS 1,235 | FOLLOWING 83 | FOLLOWERS 95 | FAVORITES 9

Follow

theconnor

@theconnor

theconnor.net

theconnor.net

Joined January 2009

@theconnor's Tweets are protected.

Only confirmed followers have access to @theconnor's Tweets and complete profile. Click the "Follow" button to send a follow request.

theconnor → Connor Riley →

What about fingerprints in source code?

zooko
@zooko

I just heard from an intern at Apple that they disallow her from contributing to open source on her own time. That's illegal, right?

RETWEETS 11 LIKES 6

11:04 AM - 4 Dec 2015

DE-ANONYMIZING PROGRAMMERS VIA CODE STYLOMETRY

De-anonymizing Programmers via Code Stylometry.
Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt.
Usenix Security Symposium, 2015

Source code stylometry

- Everyone learns coding on an individual basis, as a result code in a unique style, which makes de-anonymization possible.
- Software engineering insights
  - programmer style changes while implementing sophisticated functionality
  - differences in coding styles of programmers with different skill sets
- Identify malicious programmers.

Source code stylometry: Who wrote this code?

- **Scenario 1:**

Alice analyzes a library with malicious source code.

Bob has a source code collection with known authors.

Bob will search his collection to find Alice's adversary.

Source code stylometry: Who wrote this code?

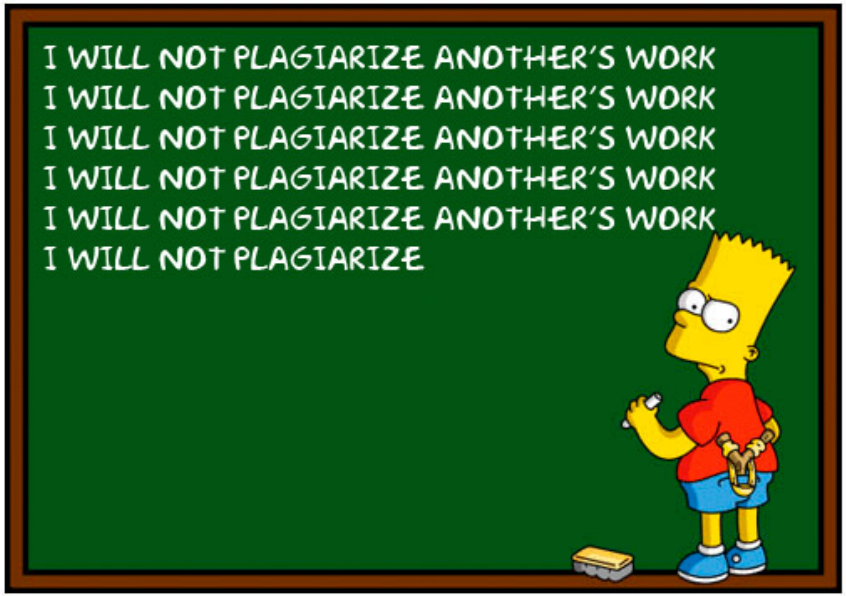
- **Scenario 2:**

Alice got an extension for her programming assignment.

Bob, the professor has

everyone else's code.

Bob wants to see if Alice plagiarized.



Source code stylometry

Iran confirms death sentence for 'porn site' web programmer

No technical difference between security-enhancing and privacy-infringing…

Comparison to related work

Related Work	Author Size	Instances	Average LOC	Language	Fetaures	Method	Result
MacDonell et al.	7	351	148	C++	lexical & layout	Case-based reasoning	88.0%
Frantzeskou et al.	8	107	145	Java	lexical & layout	Nearest neighbor	100.0%
Elenbogen and Seliya	12	83	100	C++	lexical & layout	C4.5 decision tree	74.7%
Shevertalov et. al.	20	N/A	N/A	Java	lexical & layout	Genetic algorithm	80%
Frantzeskou et al.	30	333	172	Java	lexical & layout	Nearest neighbor	96.9%
Ding and Samadzadeh	46	225	N/A	Java	lexical & layout	Nearest neighbor	75.2%
Ours	**35**	**315**	**68**	**C++**	**lexical & layout & syntactic**	**Random forest**	**100.0%**
Ours	**250**	**2,250**	**77**	**C++**			**98.0%**
Ours	**1,600**	**14,400**	**70**	**C++**			**93.6%**

Comparison to related work

Related Work	Author Size	Instances	Average LOC	Language	Fetaures	Method	Result
Frantzeskou et al.	30	333	172	Java	lexical & layout	Nearest neighbor	96.9%
Ding and Samadzadeh	46	225	N/A	Java	lexical & layout	Nearest neighbor	75.2%
Ours	35	315	68	C++	lexical & layout & syntactic	Random forest	100.0%
Ours	250	2,250	77	C++			98.0%
Ours	1,600	14,400	70	C++			93.6%

Comparison to related work

Related Work	Author Size	Instances	Average LOC	Language	Fetaures	Method	Result
Frantzeskou et al.	30	333	172	Java	lexical & layout	Nearest neighbor	96.9%
Ding and Samadzadeh	46	225	N/A	Java	lexical & layout	Nearest neighbor	75.2%
Ours	35	315	68	C++	lexical & layout & syntactic	Random forest	100.0%
Ours	250	2,250	77	C++			98.0%
Ours	1,600	14,400	70	C++			93.6%

Source code stylometry
Machine learning workflow

Dataset in CPP ~100,000 users

preprocessing

Fuzzy AST parser

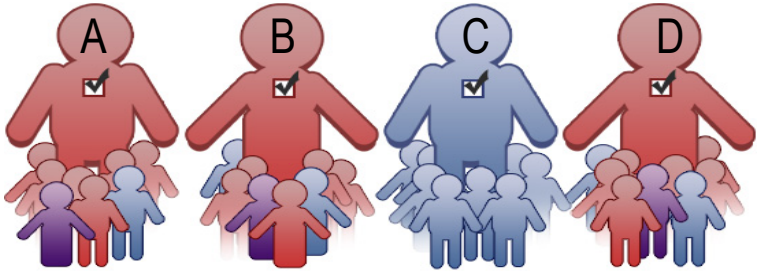
joern

Extract features

Random Forest

classification

majority vote



code jam

System.out.println("hello, world!");

- 2008-2014
- Same problems
- Limited time
- Problems get harder
- C++ most common

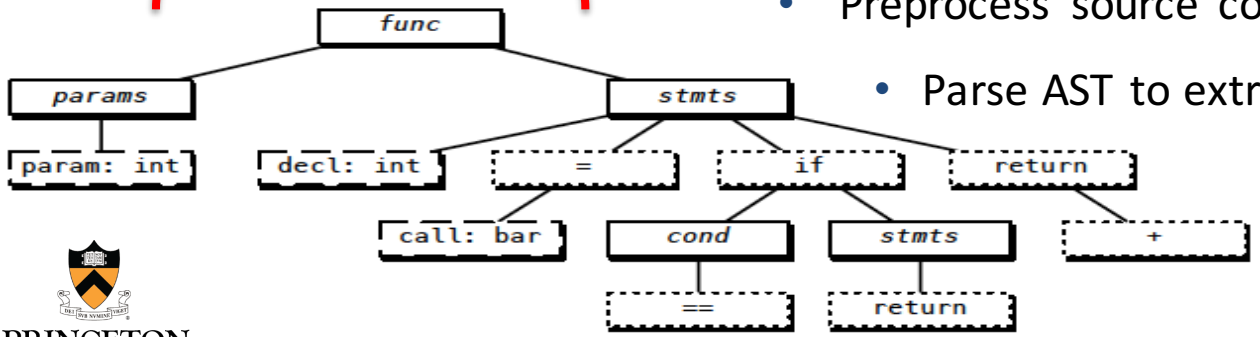
Source code stylometry

```
int foo(int y)
{
    int n = bar(y);

    if (n == 0)
        return 1;

    return (n + y);
}
```

Source Code

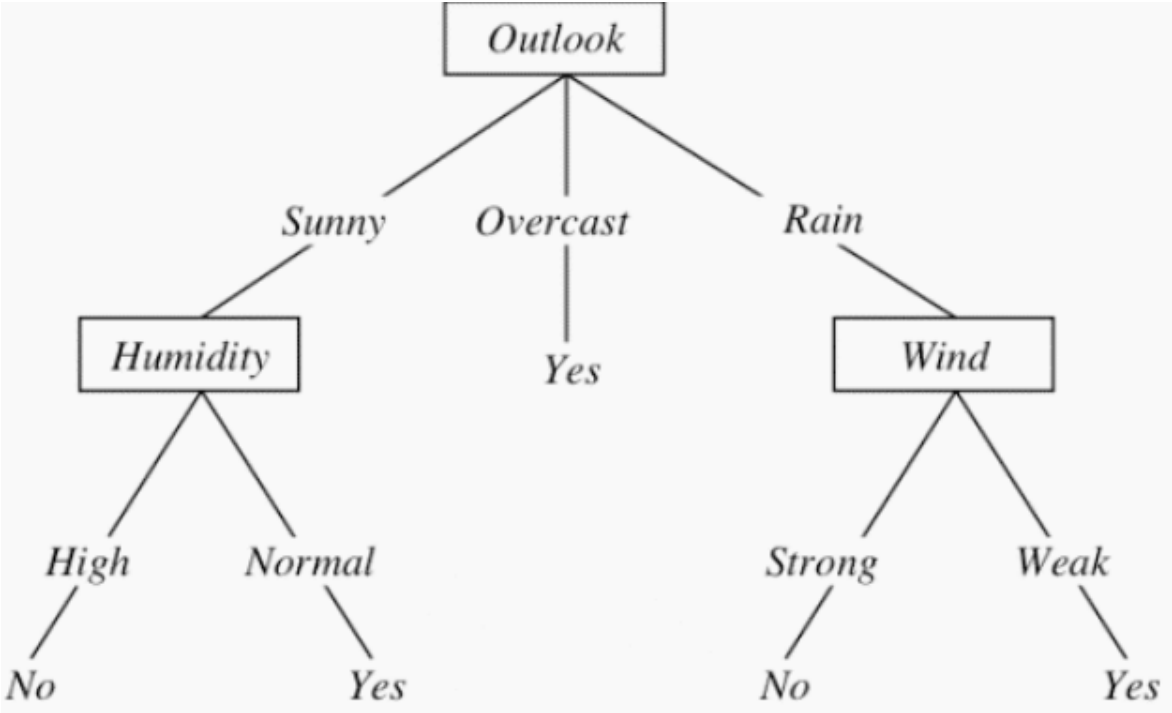
- Stylometry can be used in source code to identify the author of a program.
- Extract layout and lexical features from source code.

AST



- Abstract syntax trees (AST) in code represent the structure of the program.
  - Preprocess source code to obtain AST.
  - Parse AST to extract coding style features.

Random Forests are made of decision trees

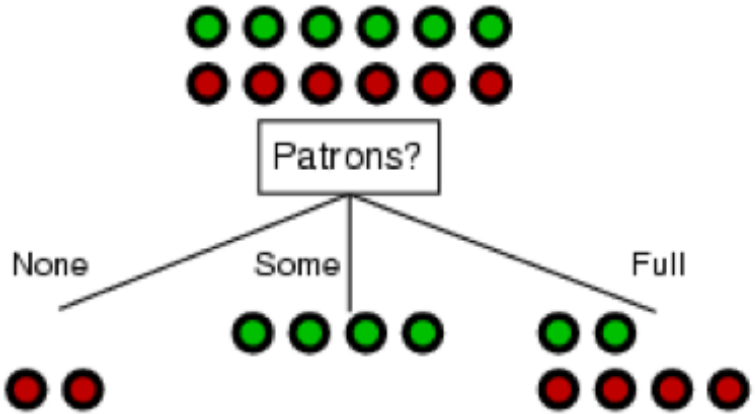


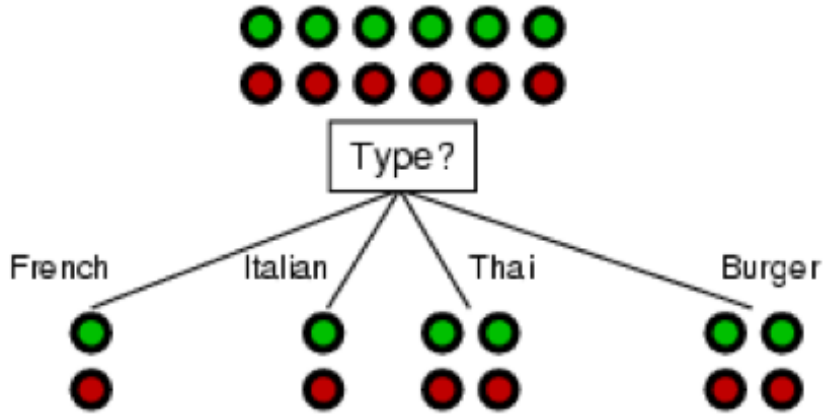
Decision Trees

- Representation
  - Each internal node tests an attribute
  - Each branch is an attribute value
  - Each leaf assigns a classification

Choosing an attribute

Which is better?





Information gain

- A chosen attribute *A* divides the training set *E* into subsets $E_1, \ldots, E_v$ according to their values for *A*, where *A* has v distinct values.

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

Information gain

For the training set, $p = n = 6$, *I(6/12, 6/12) = 1* bit

Consider the attributes *Patrons* and *Type* (and others too):

$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .0541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

934 important features of code stylometry

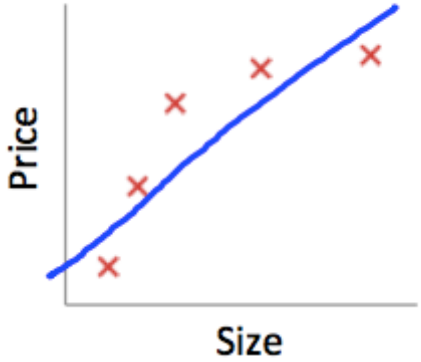
by information gain

OUT OF 120,000 FEATURES

Feature Type	Percentage	Count
Word Unigram Frequency	55%	517
AST Node-Bigram Frequency	31%	291
AST Node Average Depth	5%	48
AST Node Frequency	4%	38
AST Node TFIDF	2%	19
C++ Keywords	2%	15
Layout Features	1%	6

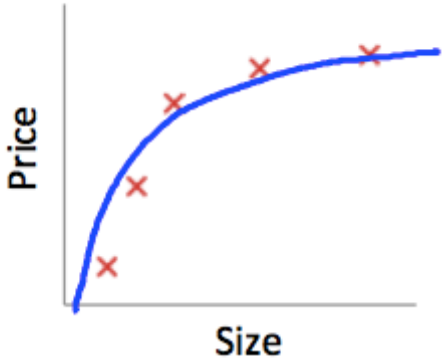
When machine learning goes wrong

- Bias vs variance



$\theta_0 + \theta_1 x$

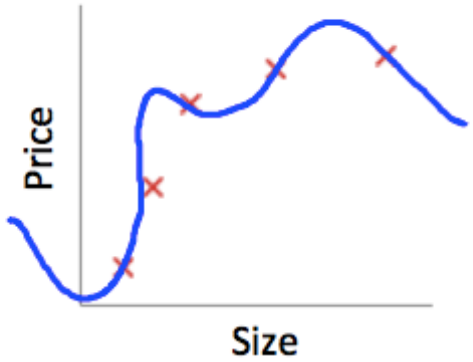
High bias (underfit)



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

$d = 2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance (overfit)

Random Forest

- Individual Trees have low bias, but high variance
- Problem: Overfitting
- Solution: Forest not a tree, build trees on subsets of the training data, using subsets of the features

Source code stylometry

Method

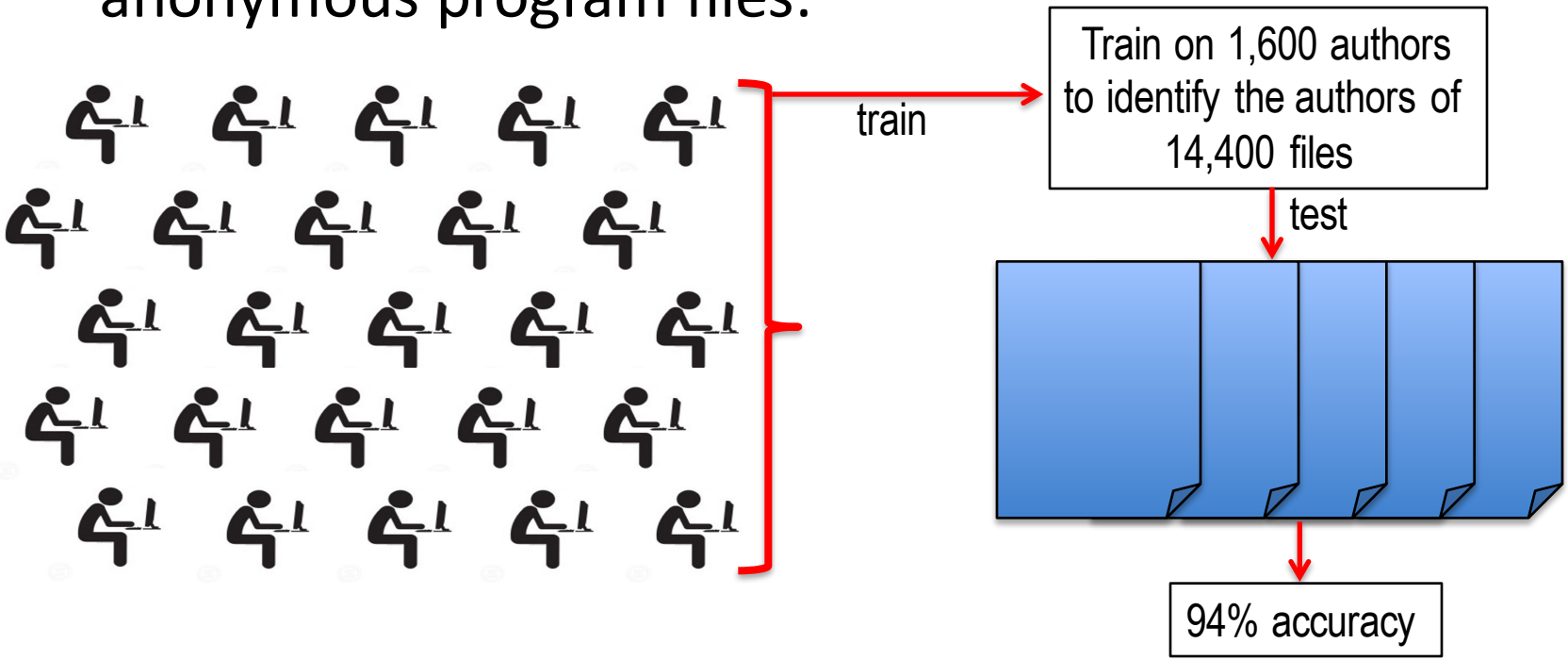
① Use random forest as the machine learning classifier,
  ① avoid over-fitting
  ② multi-class classifier by nature
② K-fold cross validation
③ Validate method on a different dataset

Case 1: Authorship attribution

- Who is this anonymous programmer?
- Who is Satoshi?

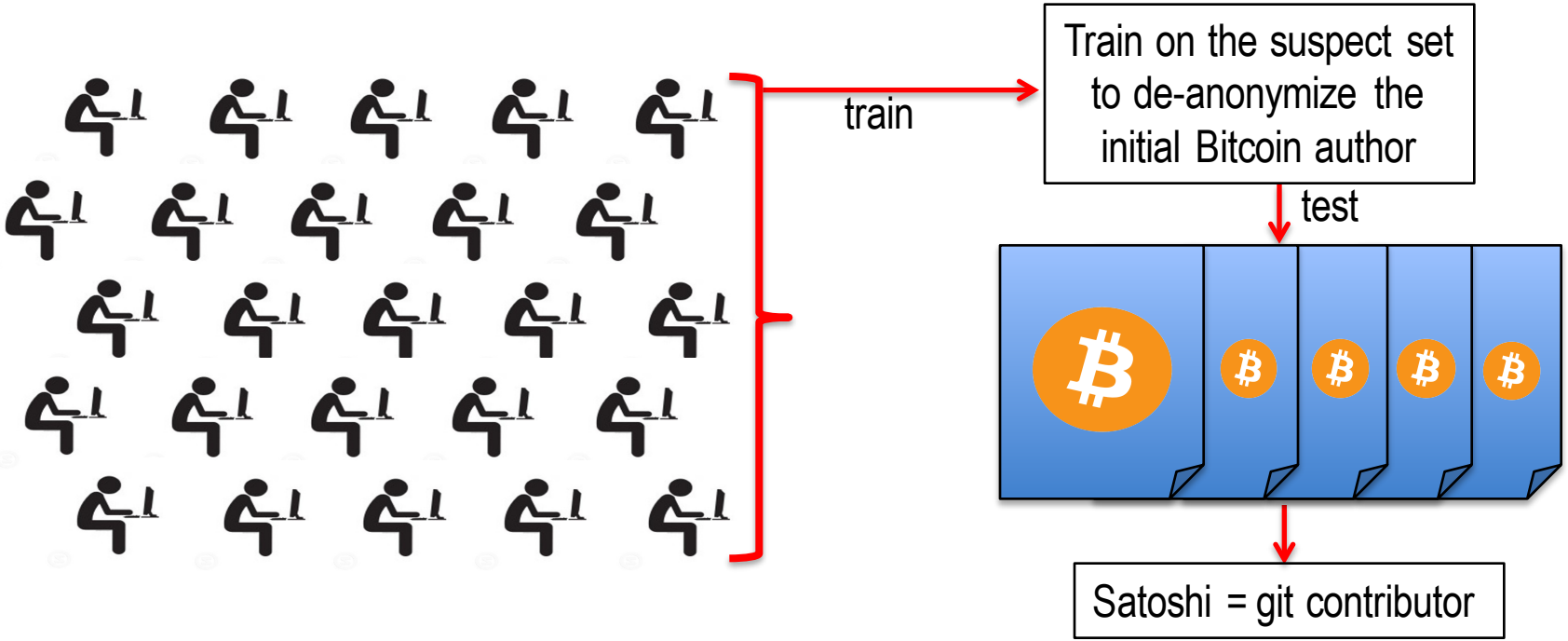
Case 1: Authorship attribution

- 94% accuracy in identifying 1,600 authors of 14,400 anonymous program files.



Case 1: Authorship attribution

- If only we had a suspect set for Satoshi…



Case 2: Obfuscation

- Who is the programmer of this obfuscated source code?
- Code is obfuscated to become unrecognizable.
- Our authorship attribution technique is impervious to common off-the-shelf source code obfuscators.

Case 2: C++ Obfuscation - STUNNIX

Sample file with C++ code

```
#ifdef __STL_USE_EXCEPTIONS /* this is conditional preprocessing */
extern void __out_of_range (const char *);
#define OUTOFRANGE(cond,msg) \
  do { if (cond) __out_of_range (#cond); } while (0)
#else
#include <cassert>
#define OUTOFRANGE(cond,msg) assert (!(cond))
#endif

template <class charT, class traits, class Allocator>
basic_string <charT, traits, Allocator>&
basic_string <charT, traits, Allocator>::
replace (size_type pos1, size_type n1,
  const basic_string& str, size_type pos2, size_type n2)
{
  //rather complex body follows
  const size_t len2 = str.length () + 2;

  if (pos1 == 0 && n1 >= length () && pos2 == 0 && n2 >= len2)
```

Case 2: C++ Obfuscation - STUNNIX

Sample file with C++ code

```
#ifdef __STL_USE_EXCEPTIONS /* this is conditional preprocessing */
extern void __out_of_range (const char *);
#define OUTOFRANGE(cond,msg) \
  do { if (cond) __out_of_range (#cond); } while (0)
#else
#include <cassert>
#define OUTOFRANGE(cond,msg) assert (!(cond))
#endif

template <class charT, class traits, class Allocator>
basic_string <charT, traits, Allocator>&
basic_string <charT, traits, Allocator>::
replace (size_type pos1, size_type n1,
  const basic_string& str, size_type pos2, size_type n2)
{
  //rather complex body follows
  const size_t len2 = str.length () + 2;

  if (pos1 == 0 && n1 >= length () && pos2 == 0 && n2 >= len2)
```

Case 2: C++ Obfuscation - STUNNIX

Sample file with C++ code

```
#ifdef z7929401884
extern void za41dafc42e(const char*);
#define z1c52ffdd48(z22fc207d33,zde05b8b1b0) \
  do { if (z22fc207d33) za41dafc42e (#z22fc207d33); } while ((0x1a1+8313-0x221a))
#else
#include <cassert>
#define z1c52ffdd48(z22fc207d3
#endif
template<class zd9cfc9cefe,cla
zd9cfc9cefe,z9cdf2cd536,Alloca
::replace(size_type z795f772c7
size_type z8ad17de27a,size_type za2e5f06cde){
const size_t z51dea41ale=str.length()+(0x12ac+3131-0x1ee5);if(z795f772c7c==
(0x455+8190-0x2453)&&zddd43c876a>=length()&&z8ad17de27a==(0xc15+4853-0x1f0a)&&
za2e5f06cde>=z51dea41ale)return operator=(str);z1c52ffdd48(z8ad17de27a>
z51dea41ale,"\x65\x72\x72\x6f\x72\x20\x69\x6e\x20\x72\x65\x70\x6c\x61\x63\x65");
#ifdef zd943335d79
++::z021c346d26.z1534cdbaf9;
#endif
```

Same set of 20 authors with 180 program files	Classification Accuracy
Original source code	99%
STUNNIX-Obfuscated source code	99%

Case 2: C Obfuscation - TIGRESS

```c
#include<stdio.h>
int main()
{
  int T,test=1;
  double C,F,X,rate,time;
  scanf("%d",&T);
  while(T--)
    {
      scanf("%lf %lf %lf",&C,&F,&X);
      rate=2.0;
      time=0;
      while(X/rate>C/rate+X/(rate+F))
    {
      time+=C/rate;
      rate+=F;
    }
      time+=X/rate;
      printf("Case #%d: %lf\n",test++,time);
    }
  return 0;
}
```

Case 2: C Obfuscation - TIGRESS

```c
#include<stdio.h>
int main()
{
  int T,test=1;
  double C,F,X,rate,time;
  scanf("%d",&T);
  while(T--)
    {
      scanf("%lf %lf %lf",&C,&F,&X);
      rate=2.0;
      time=0;
      while(X/rate>C/rate+X/(rate+F))
    {
      time+=C/rate;
      rate+=F;
    }
      time+=X/rate;
      printf("Case #%d: %lf\n",test++,time);
    }
  return 0;
}
```

```
struct _IO_FILE;
struct timeval {
   long tv_sec ;
   long tv_usec ;
};
enum _1_main_$op {
    _1_main__string
$value_LIT_0$result_REG_1__convert_void_star2void_star
$result_STA_0$left_REG_0__local
$result_STA_0$value_LIT_0__store_void_star
$left_STA_0$right_STA_1__local
$result_STA_0$value_LIT_0__convert_void_star2void_star
$left_STA_0$result_REG_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
$result_STA_0$left_REG_0__store_void_star$right_STA_0$left_REG_0 =
46,
    _1_main__local$result_REG_0$value_LIT_1__constant_int
$result_STA_0$value_LIT_0__store_int$right_STA_0$left_REG_0__local
$result_STA_0$value_LIT_0__convert_void_star2void_star
$left_STA_0$result_REG_0__string
$value_LIT_0$result_REG_1__convert_void_star2void_star
$result_STA_0$left_REG_0__store_void_star
$right_STA_0$left_REG_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
$result_STA_0$left_REG_0 = 44,
    _1_main__convert_void_star2void_star
$left_STA_0$result_REG_0__load_int
$left_REG_0$result_REG_1__MinusA_int_int2int
$result_REG_0$left_REG_1$right_REG_2__store_int
$left_STA_0$right_REG_0__goto$label_LAB_0 = 161,
    _1_main__local$result_STA_0$value_LIT_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
$result_STA_0$left_REG_0__load_double
$left_STA_0$result_REG_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
$result_STA_0$left_REG_0__load_double
$left_STA_0$result_STA_0__convert_double2double
$left_STA_0$result_REG_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
```

Case 2: C Obfuscation - TIGRESS

```
#include<stdio.h>
int main()
{
  int T,test=1;
  double C,F,X,rate,time;
  scanf("%d",&T);
  while(T--)
    {
      scanf("%lf %lf %lf",&C
      rate=2.0;
      time=0;
      while(X/rate>C/rate+X/
    {
      time+=C/rate;
      rate+=F;
    }
      time+=X/rate;
      printf("Case #%d: %lf
    }
  return 0;
}
```

```
struct _IO_FILE;
struct timeval {
   long tv_sec ;
   long tv_usec ;
};
enum _1_main_$op {
    _1_main__string
$value_LIT_0$result_REG_1__convert_void_star2void_star
$result_STA_0$left_REG_0__local
$result_STA_0$value_LIT_0__store_void_star
```

Same set of 20 authors with 180 program files	Classification Accuracy
Original C source code	96%
TIGRESS-Obfuscated source code	67%
Random chance of correct de-anonymization	5%

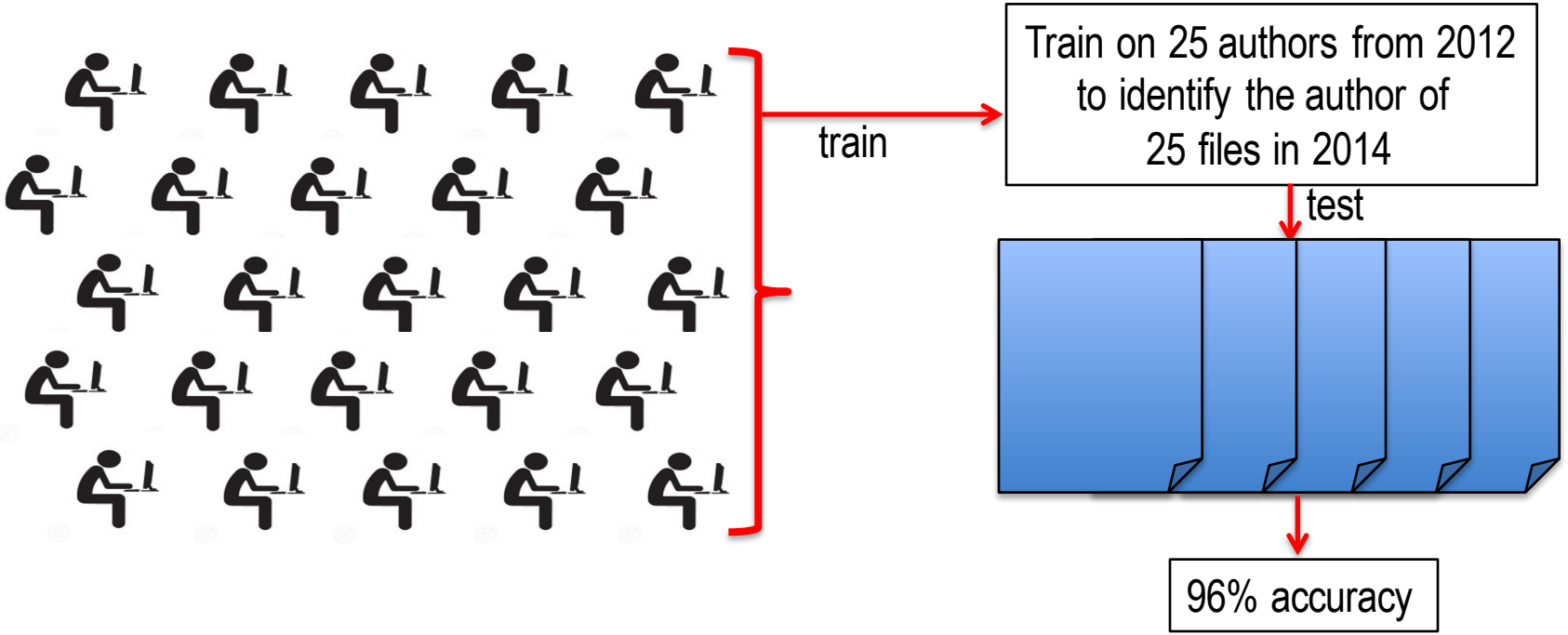
```
eft_REG_0 =
nt
EG_0__local
$left_REG_0$result_REG_1__MinusA_int_int2int
$result_REG_0$left_REG_1$right_REG_2__store_int
$left_STA_0$right_REG_0__goto$label_LAB_0 = 161,
    _1_main__local$result_STA_0$value_LIT_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
$result_STA_0$left_REG_0__load_double
$left_STA_0$result_REG_0__local
$result_REG_0$value_LIT_1__convert_void_star2void_star
$result_STA_0$left_REG_0__load_double
$left_STA_0$result_STA_0__convert_double2double
$left_STA_0$result_REG_0__local
```

Case 3: Coding style throughout years

- Is programming style consistent?
- If yes, we can use code from different years for authorship attribution.

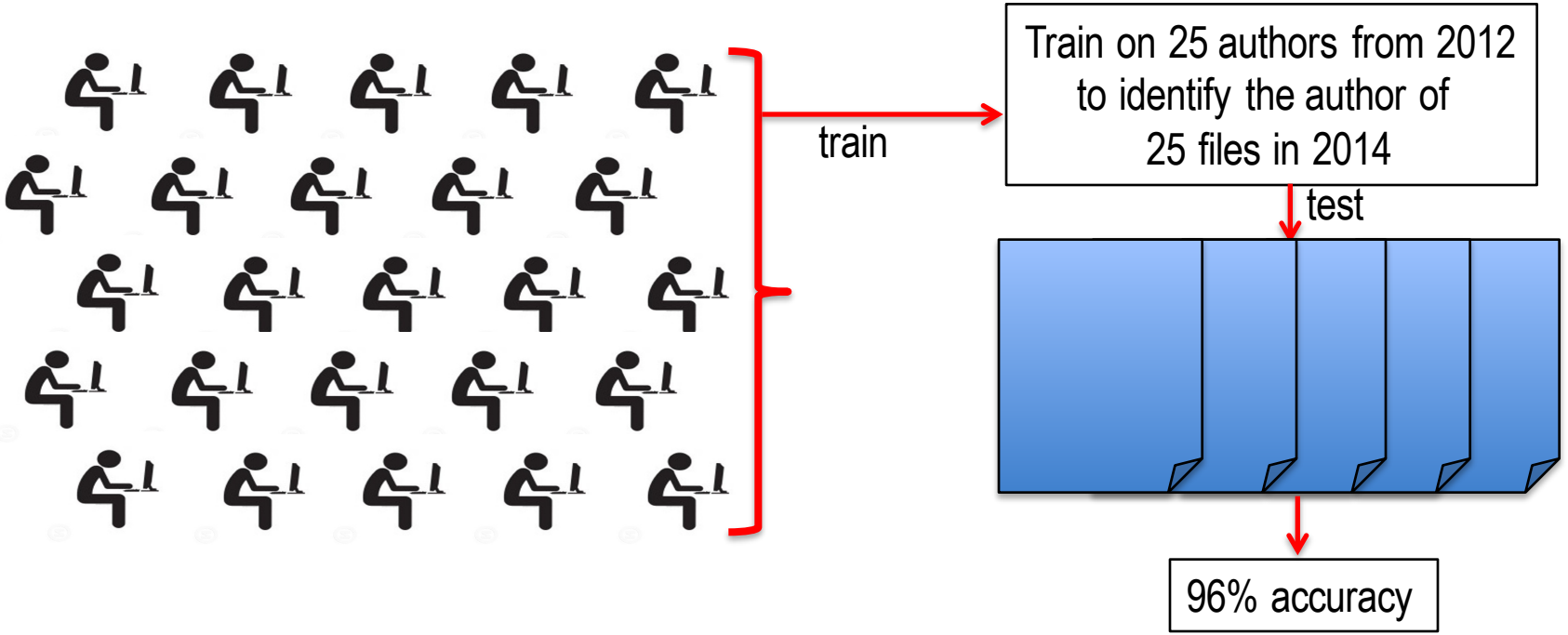
Case 3: Coding style throughout years

- Coding style is preserved up to some degree throughout years.



Case 3: Coding style throughout years

- 98% accuracy, train and test in 2014
- 96% accuracy, train on 2012, test on 2014



Case 4: Generalizing the approach - python

Feature set: Using ‘only’ the Python equivalents of syntactic features

Application	Programmers	Instances	Result
Python programmer de-anonymization	229	2,061	53.9%
Top-5 relaxed classification	229	2,061	75.7%
Python programmer de-anonymization	23	207	87.9%
Top-5 relaxed classification	23	207	99.5%

Results

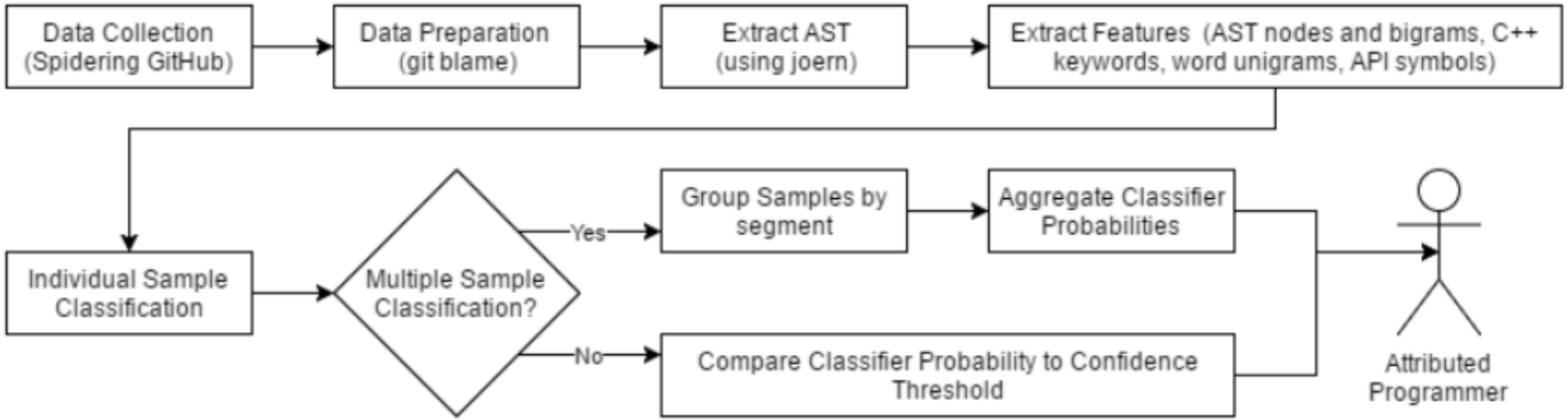
A new principled method with a robust syntactic feature set for de-anonymizing programmers.

Application	Classes	Instances	Accuracy
Stylometric plagiarism detection	250 class	2,250	98.0%
Large scale de-anonymization	1,600 class	14,400	93.6%
Copyright investigation	Two-class	1,080	100.0%
Authorship verification	Two-class/One-class	2,240	91.0%
Open world problem	Multi-class	420	96.0%

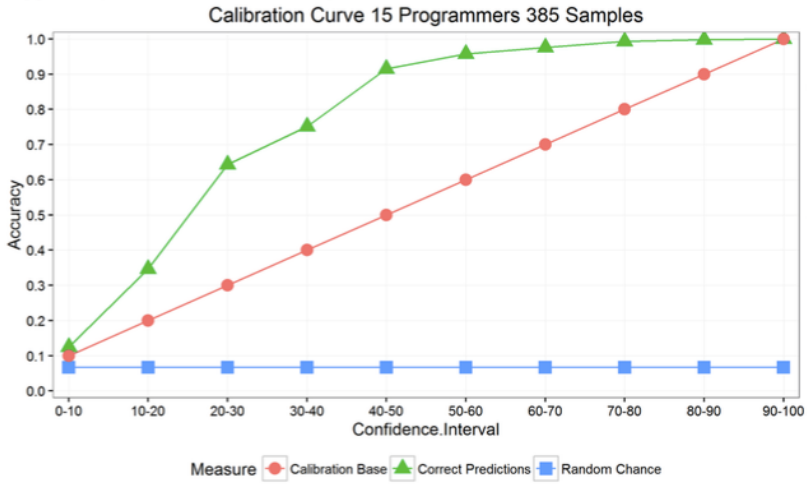
Git Blame Who?

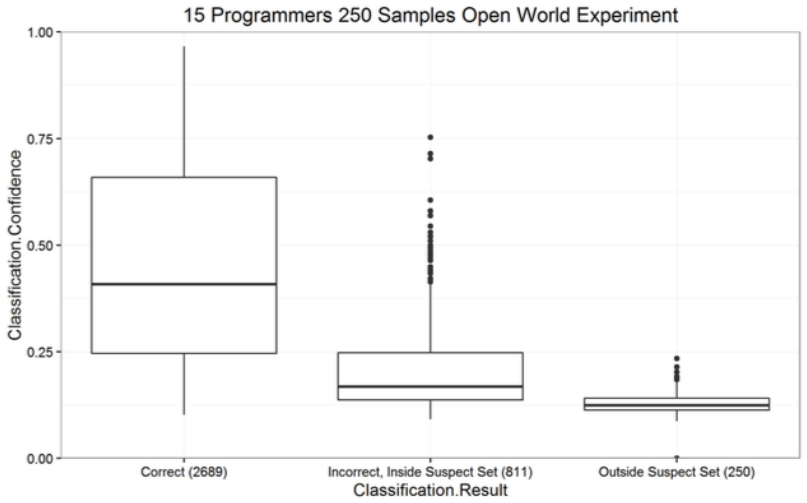
- A lot of code is collaborative
- > 70% accuracy for individual attribution for a single git commit, higher for multiple commits/account
- For accounts, we can attribute single commits and ensemble them (errors are uncorrelated, accuracy quickly reaches close to 100%)

Attributing Code Fragments

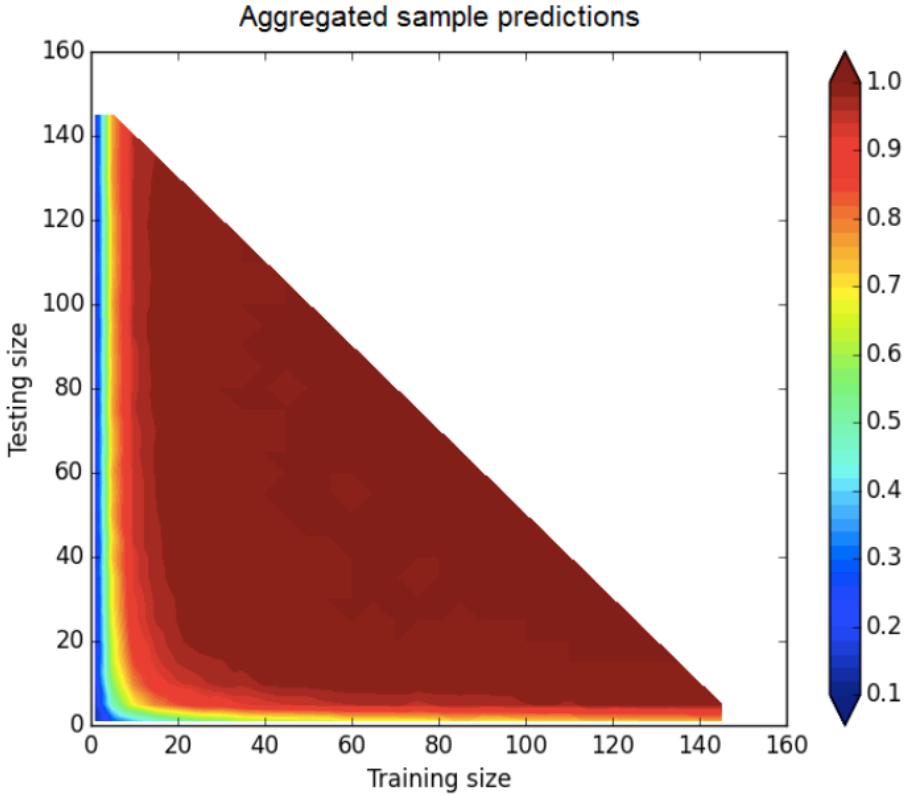


Open World Attribution





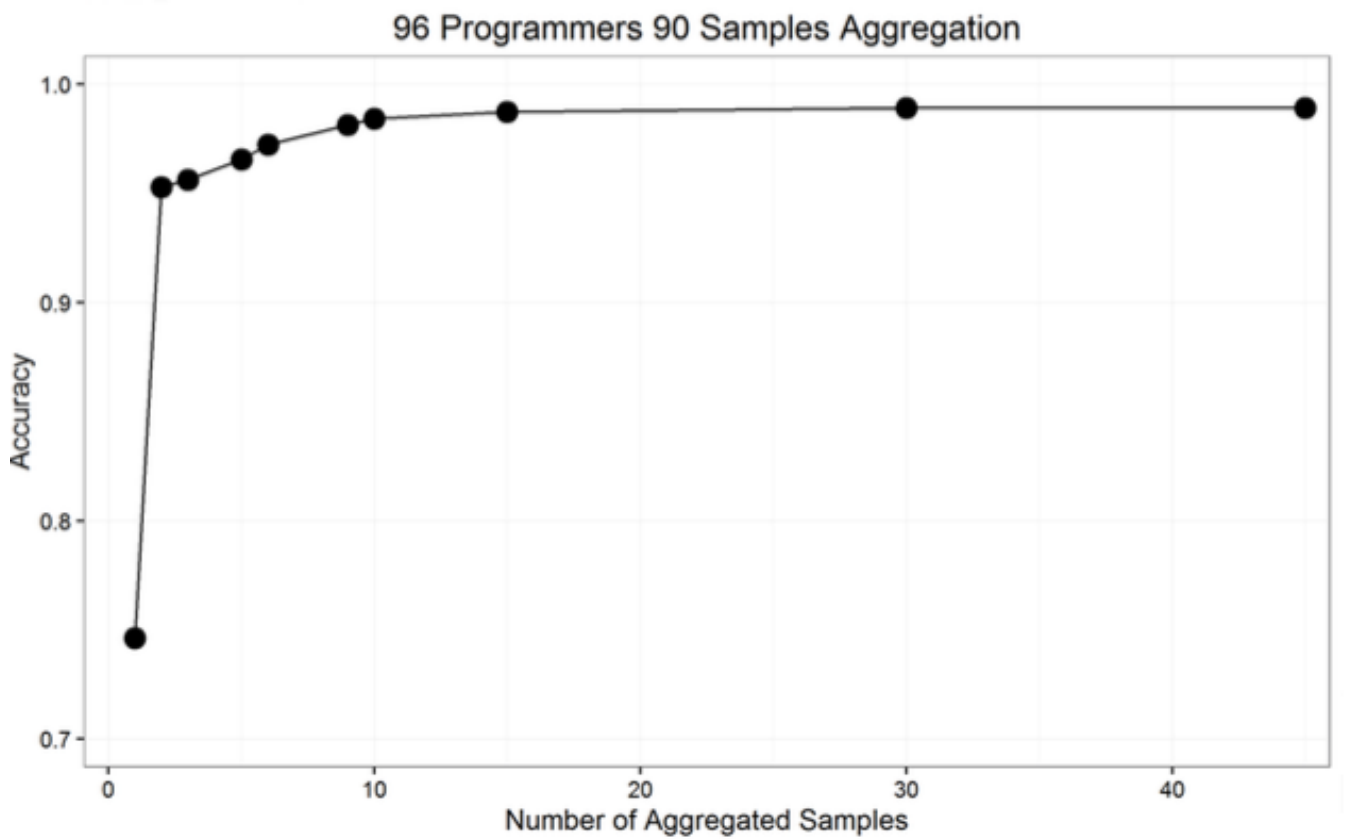
How much data for attribution?



How much code?

Samples	min LOC	Programmers	Accuracy
4	38	90	54%
6	28	90	63%
10	18	90	76%
23	8	90	75%
90	3	90	77%
150	1	90	75%

Aggregated Samples



What about executable binaries?

Source Code → Compiled code looks cryptic

```
#include <cstdio>
#include <algorithm>
using namespace std;
#define For(i,a,b) for(int i = a; i < b; i++)
#define FOR(i,a,b) for(int i = b-1; i >= a; i--)
double nextDouble() {
        double x;
        scanf("%lf", &x);
        return x;}
int nextInt() {
        int x;
        scanf("%d", &x);
        return x; }
int n;
double a1[1001], a2[1001];
int main() {
        freopen("D-small-attempt0.in", "r", stdin);
        freopen("D-small.out", "w", stdout);
        int tt = nextInt();
        For(t,1,tt+1) {
                int n = nextInt();
        . . .
```

```
00100000 00000000 00001000 00000000 00101000 00000000
00000000 00000000 00110100 00000000 00000000 00000000
00000100 00001000 00000000 00000001 00000000 00000000
00000000 00000001 00000000 00000000 00000101 00000000
00000000 00000000 00000100 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00110100 00000001
00000000 00000000 00110100 10000001 00000100 00001000
00000000 00000000 00010011 00000000 00000000 00000000
00000100 00000000 00000000 00000000 00000001 00000000
00000000 00000000 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 10000000
00000100 00001000 00000000 10000000 00000100 00001000
11001000 00010111 00000000 00000000 11001000 00010111
00000000 00000000 00000101 00000000 00000000 00000000
00000000 00010000 00000000 00000000 00000001 00000000
00000000 00000000 11001000 00010111 00000000 00000000
11001000 10100111 00000100 00001000 11001000 10100111
00000100 00001000 00101100 00000001 00000000 00000000
00000000 00000000 00000000 00010000 00000000 00000000
00000010 00000000 00000000 00000000 11011100 00010111
. . .
```

Can we identify the author of this binary?

00100000 00000000 00001000 00000000 00101000 00000000
00000000 00000000 00110100 00000000 00000000 00000000
00000100 00001000 00000000 00000001 00000000 00000000
00000000 00000001 00000000 00000000 00000101 00000000
00000000 00000000 00000100 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00110100 00000001
00000000 00000000 00110100 10000001 00000100 00001000
00000000 00000000 00010011 00000000 00000000 00000000
00000100 00000000 00000000 00000000 00000001 00000000
00000000 00000000 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 10000000
00000100 00001000 00000000 10000000 00000100 00001000
11001000 00010111 00000000 00000000 11001000 00010111
00000000 00000000 00000101 00000000 00000000 00000000
00000000 00010000 00000000 00000000 00000001 00000000
00000000 00000000 11001000 00010111 00000000 00000000
11001000 10100111 00000100 00001000 11001000 10100111
00000100 00001000 00101100 00000001 00000000 00000000
00000000 00000000 00000000 00010000 00000000 00000000
00000010 00000000 00000000 00000000 11011100 00010111
...

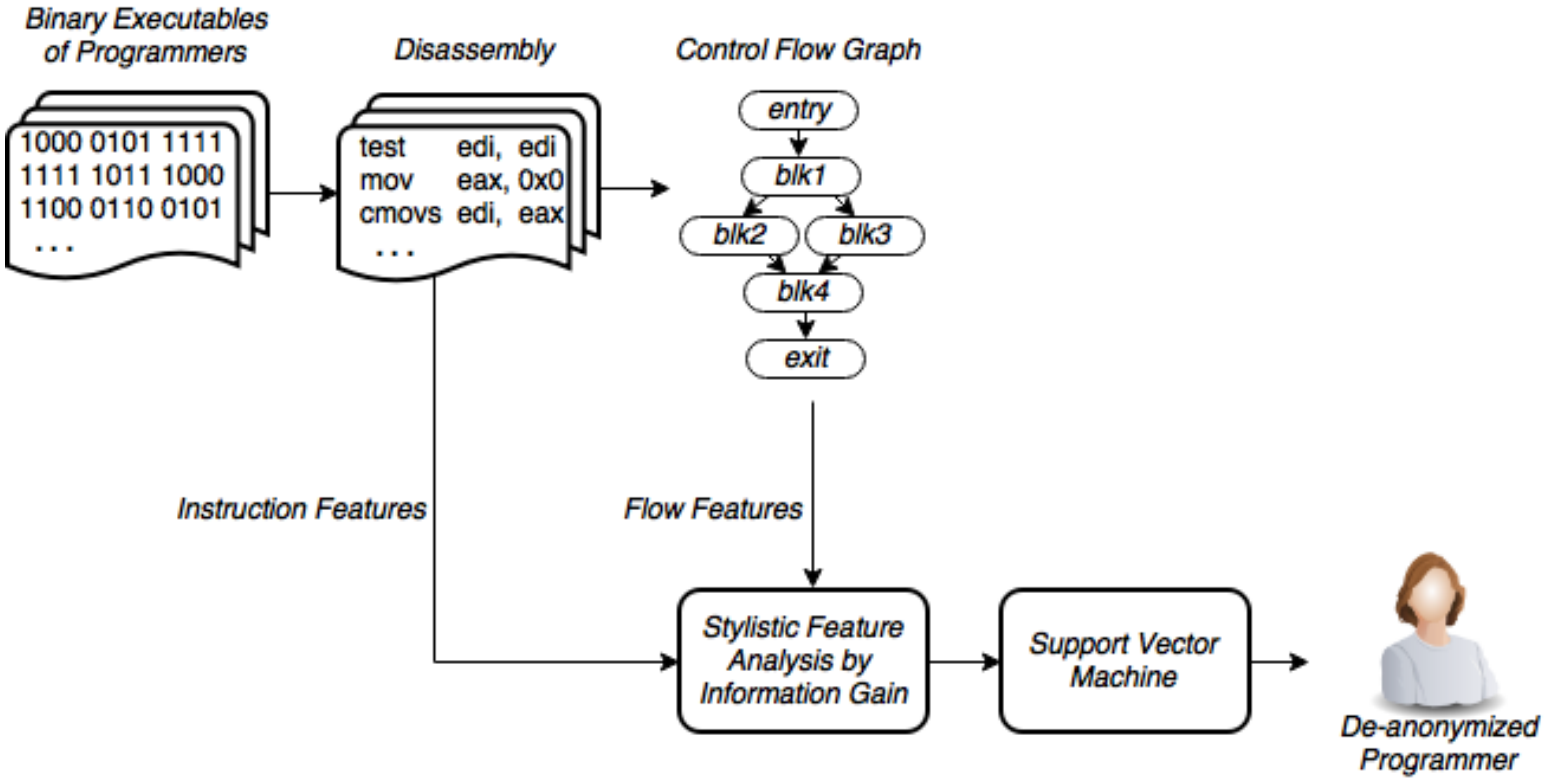
WHEN CODING STYLE SURVIVES COMPILATION: DE-ANONYMIZING PROGRAMMERS FROM EXECUTABLE BINARIES

When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries
Aylin Caliskan-Islam, Fabian Yamaguchi, Edwin Dauber, Richard Harang, Konrad Rieck, Rachel Greenstadt, and Arvind Narayanan.
Under Submission, 2016 available on arXiv

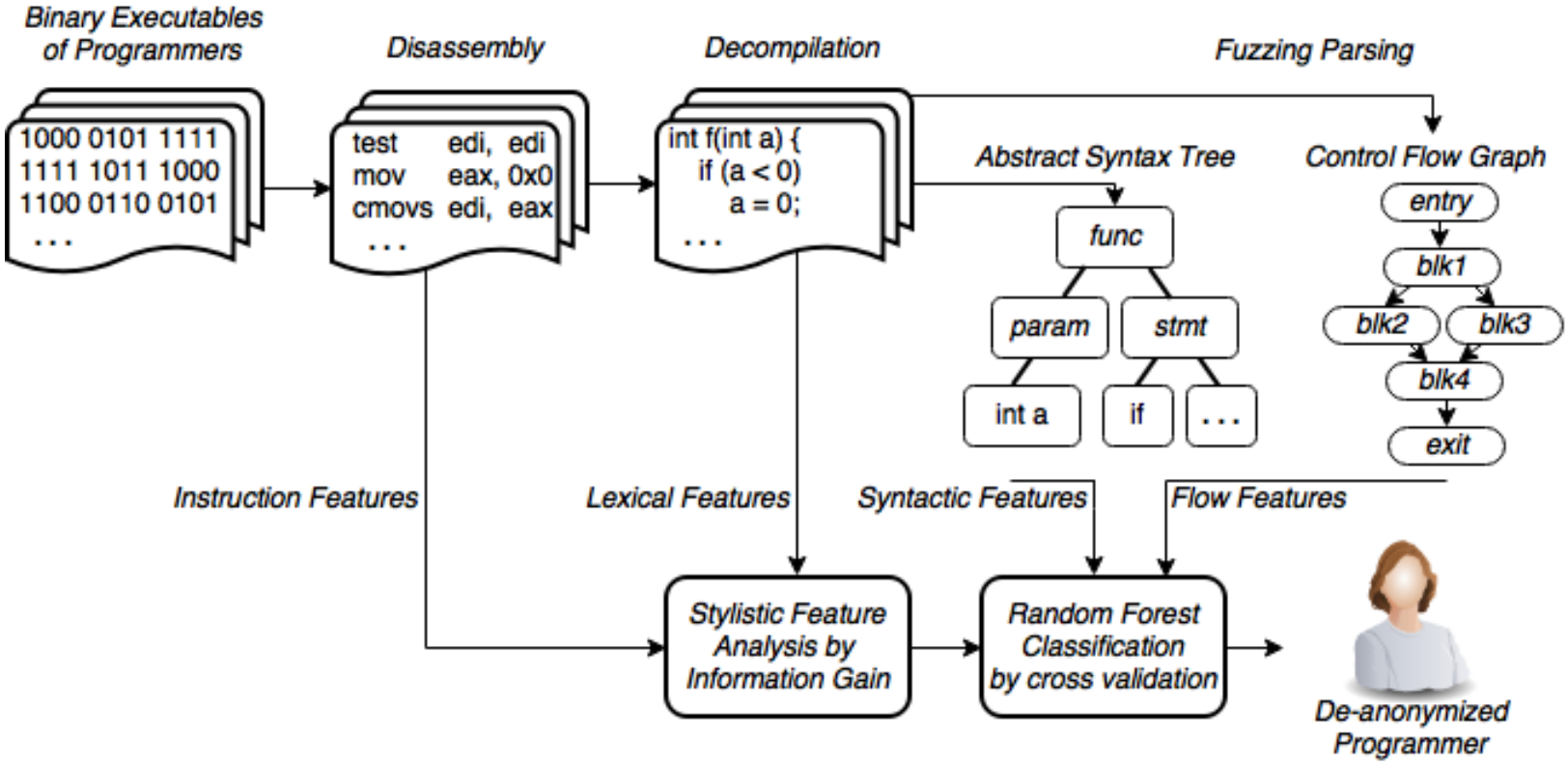
Finding the author of an executable binary?

- Coding style in compiled code
- Threat to privacy and anonymity
- Malware classification?

Related work

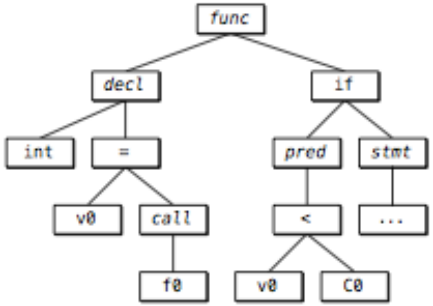


Our workflow



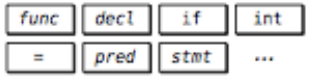
Features

Abstract syntax tree (AST)

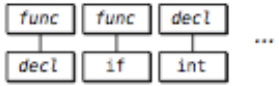


Syntactic features

AST unigrams:

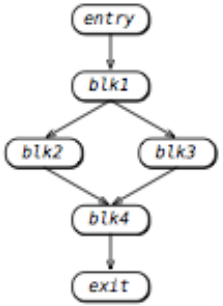


AST bigrams:



AST depth: 5

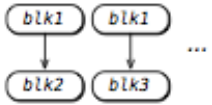
Control-flow graph (CFG)



Control-flow features

CFG unigrams:

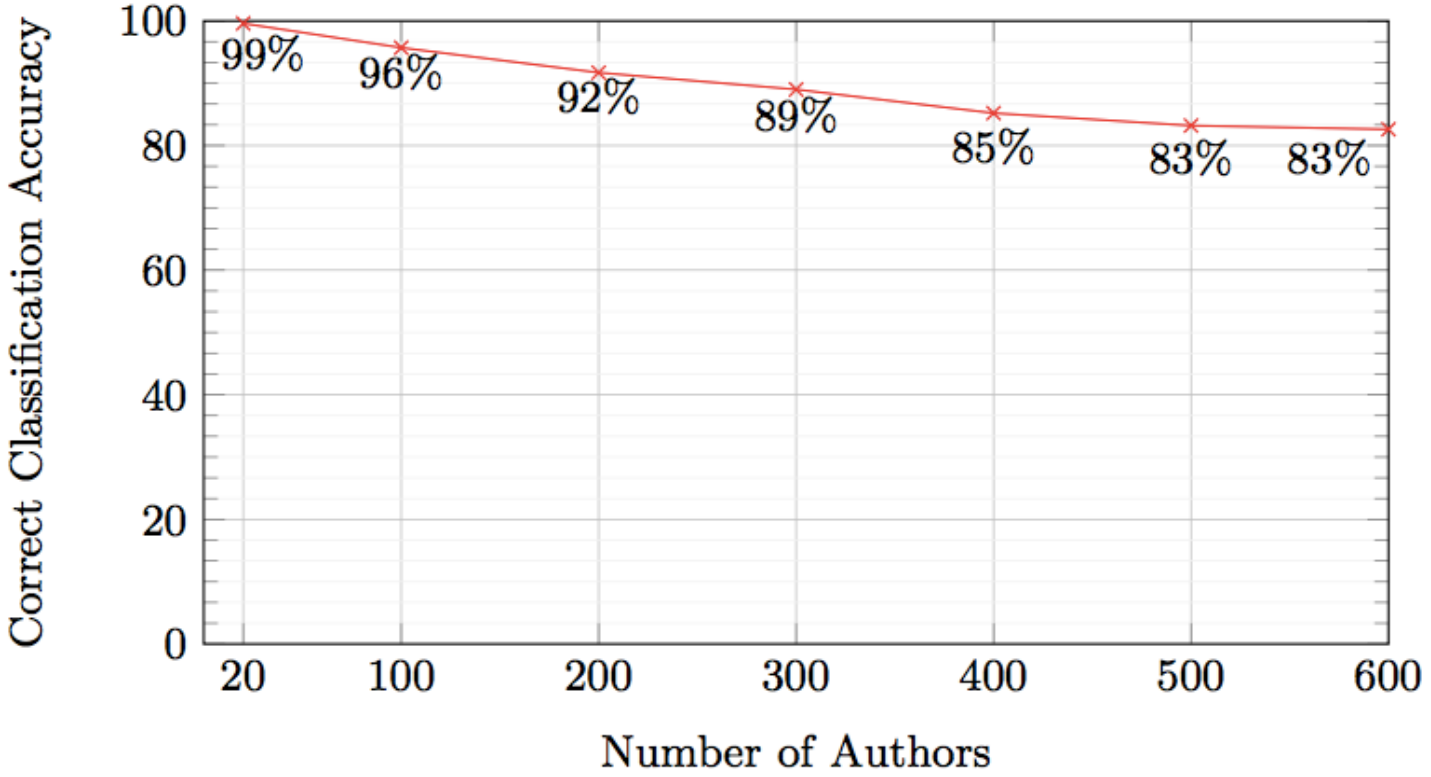
CFG bigrams:



Feature set from 100 programmers

Feature	Source	Number of Possible Features	Information Gain Features
word unigrams	hex-rays decompiled code	29,686	102
AST node TF*	hex-rays decompiled code	14,663	24
Labeled AST edge TF*	decompiled code	25,941	88
AST node TFIDF**	decompiled code	14,663	8
AST node average depth	decompiled code	14,663	21
C++ keywords	decompiled code	73	5
radare2 disassembly unigrams	radare disassembly	12,629	45
radare2 disassembly bigrams	radare disassembly	33,919	75
ndisasm disassembly unigrams	ndisasm disassembly	532	8
ndisasm disassembly bigrams	ndisasm disassembly	4,570	25
CFG unigrams	Snowman CFG	11,503	5
CFG unigram TFIDF**	Snowman CFG	11,503	10
CFG bigrams	Snowman CFG	38,554	10
Total		201,396	426
TF = term frequency*			
*TFIDF** = term frequency inverse document frequency*			

Large Scale Programmer Deanonymization



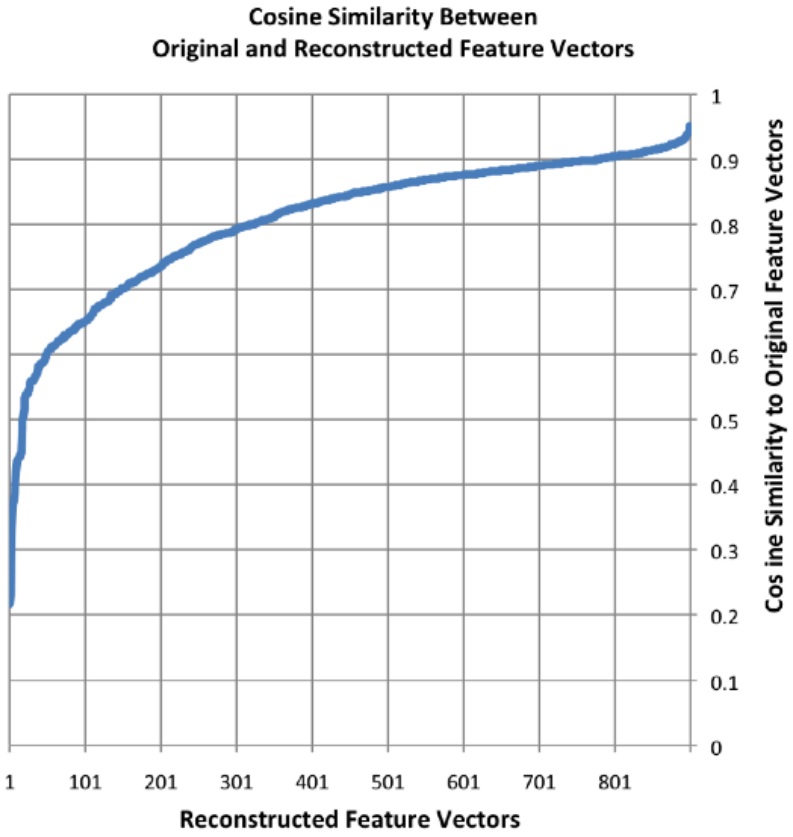
Related Work	Number of Programmers	Number of Training Samples	Accuracy	Classifier
Rosenblum [37]	20	**8-16**	77%	SVM
This work	20	**8**	90%	SVM
This work	20	**8**	99%	RF
Rosenblum [37]	100	8-16	**61%**	SVM
This work	100	8	**84%**	SVM
This work	100	8	**96%**	RF
Rosenblum [37]	191	8-16	**51%**	SVM
This work	191	8	**81%**	SVM
This work	191	8	**92%**	RF
This work	**600**	8	**71%**	SVM
This work	**600**	8	**83%**	RF

Compiler Optimization

Number of Programmers	Number of Training Samples	Compiler Optimization Level	Accuracy
100	8	None	96%
100	8	1	93%
100	8	2	89%
100	8	3	89%

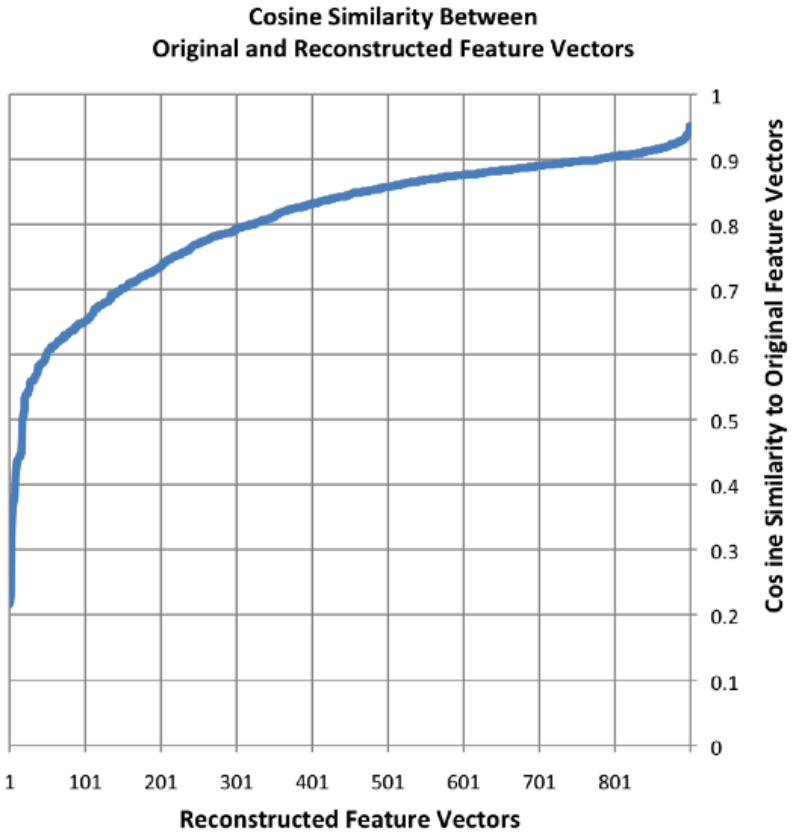
The drop in accuracy is not tragic!

Reconstructing original features



- Original vs predicted features
  - Average cos distance: 0.81
- Original vs decompiled features
  - Average cos distance: 0.35

Reconstructing original features



- Original vs predicted features
  - Average cos distance: 0.81

- *This suggests that original features are transformed but not entirely lost in compilation.*

Insights

More advanced programmers are easier to de-anonymize

A = #authors, F = max #problems completed							
N = #problems included in dataset (N ≤ F)							
A = 20							
F = 14		**F = 7**		**F = 12**		**F = 6**	
	N = 7 easier	**N = 7**			**N = 6 easier**	**N = 6**	
Average accuracy after 10 iterations							
88.2%		80.7%[1]		86.7%		78.1%[1]	
[1]**Drop in accuracy due to programmer skill set.**							

Real World Binary Attribution

- Code repositories from GitHub
  - 65% accuracy (less code to train on)
- Binaries mined from leaked Nulled.io forum
  - 4 users, 3 with enough data to train a model. 3 correctly identified, 4th identified as not one of the other 3.

Future work

- Anonymizing executable binaries
  - optimizations is not the answer
- De-anonymizing collaborative binaries
- Malware family classification

Available tools

- Programmer de-anonymization
  - https://github.com/calaylin
- Jstylo
  - prose authorship attribution framework
- Anonymouth
  - writing anonymization

Dr. Richard Harang

Dr. Konrad Rieck

Dr. Arvind Narayanan

THANKS CO-AUTHORS Edwin Dauber and

Dr. Clare Voss

Dr. Fabian Yamaguchi

Dr. Aylin Caliskan-Islam