Hunting Malware

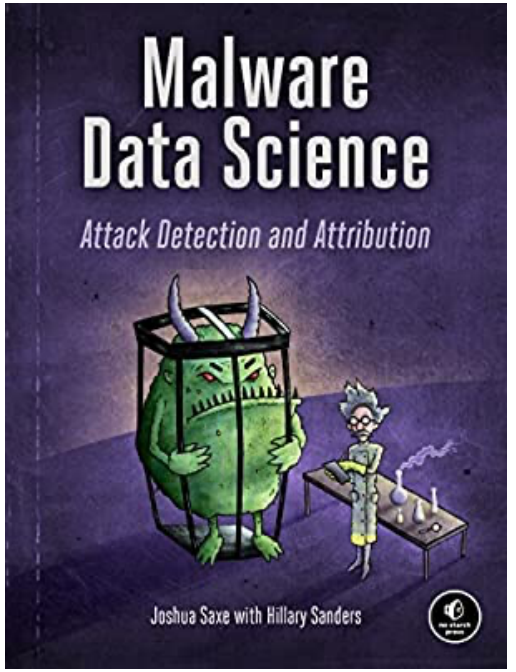
with Data Science

Who Am I?

- Robin Dimyanoglu
- SCO @ Cyber Struggle
- Low level voodo || Dark art of reversing
- Twitter: @1ce7ea

This presentation is about

- A summary of my learning journey
- Machine Learning & Malware <3



This presentation is not about

- Malware analysis / reverse engineering
- Next-Gen AI MAL-WARE SLAYER 1500 ©
  (powered by blockchain)

Table of Contents

What is Machine Learning

- Computer algorithms that improve through experience
- Actually you don’t build algorithms in order to address a problem
- Instead you build **mathematical models** based on some **data**
- These algorithms address specific problems
  - Classification
  - Forecasting
  - Pattern Recognition
  - Clustering

OK.. But how does it work?

1. **Exactly** identify your problem
2. Find (or create, or buy) a dataset related to your problem
3. Work on your dataset for a while to increase its quality and usability
4. Select which features of the data will you use to build your model
5. Use one of the algos to build the model with features you have chosen
6. Test the accuracy of your model; go back to step 4 if results are below expectation

How do I know which features to use?

Rule 1: Features have to be statistically significant

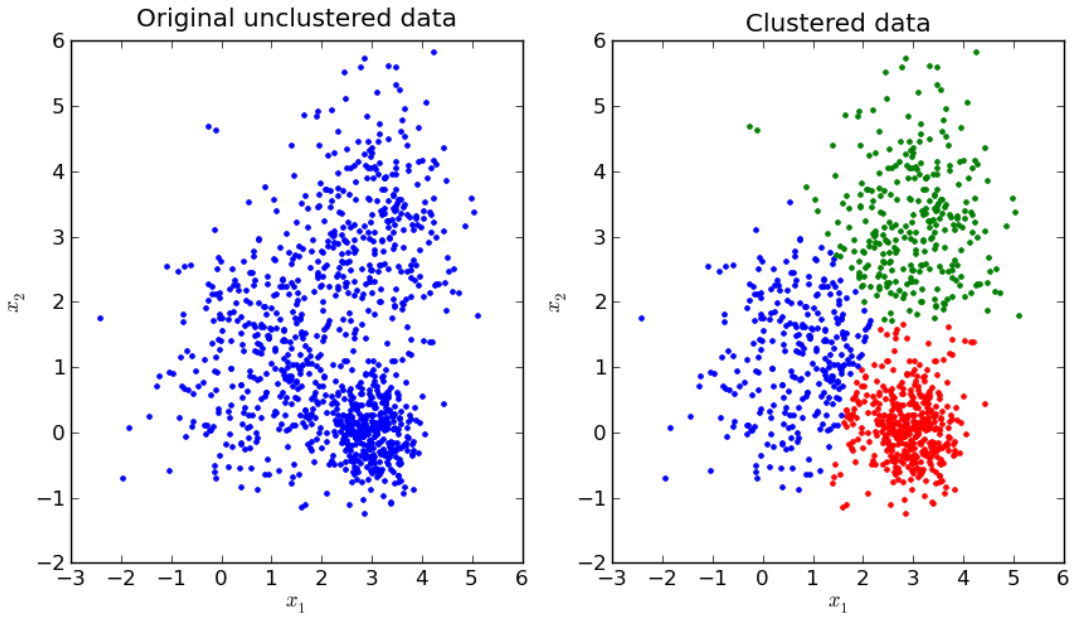
Rule 2: Features better be easily extractable

Rule 3: Lesser the features you use; the better your model will be (Curse of Dimensionality)

- Construct a hypothesis about your problem
- Visualize data in order to see the effects of individual features
- Trial and error ☺
- Use one of the Dimensionality Reduction methods

What is cluster analysis?

- Basically finding what's similar to each other in a set of data
- Useful when you don't know the classes beforehand
- Extract Features > Cluster Dataset > Test Accuracy
- Clustering means unsupervised



Why should I care?

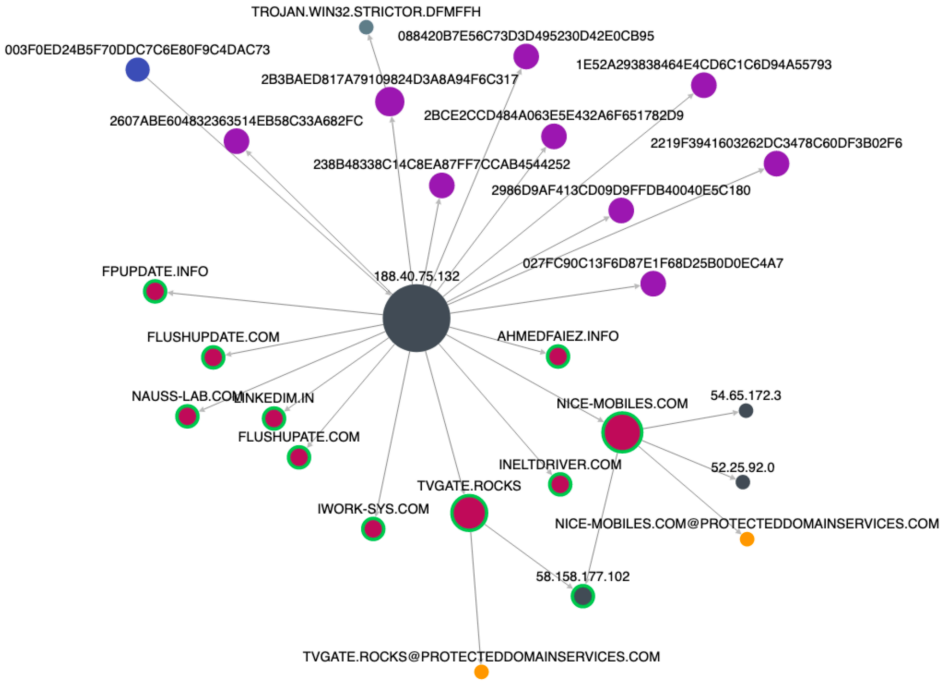
With clustering it is easier to:

- Identify previously unknown malware
- Track malware campaigns
- Find related malware samples

You may not have $$$ to buy enterprise sandbox

Traditional Methods for Mal ID

- Configuration based
  - Hypothesis: if two malware uses common infras (for CnC) they are related
- **Different** malware could use **same** CnC
- **Same** malware could use **different** CnC



Traditional Methods for Mal ID

- Signature based
  - Hypothesis: if two malware share the same unique property they are related
- Static signatures are usually strings and properties of the file
- But code portions can be used as signatures too

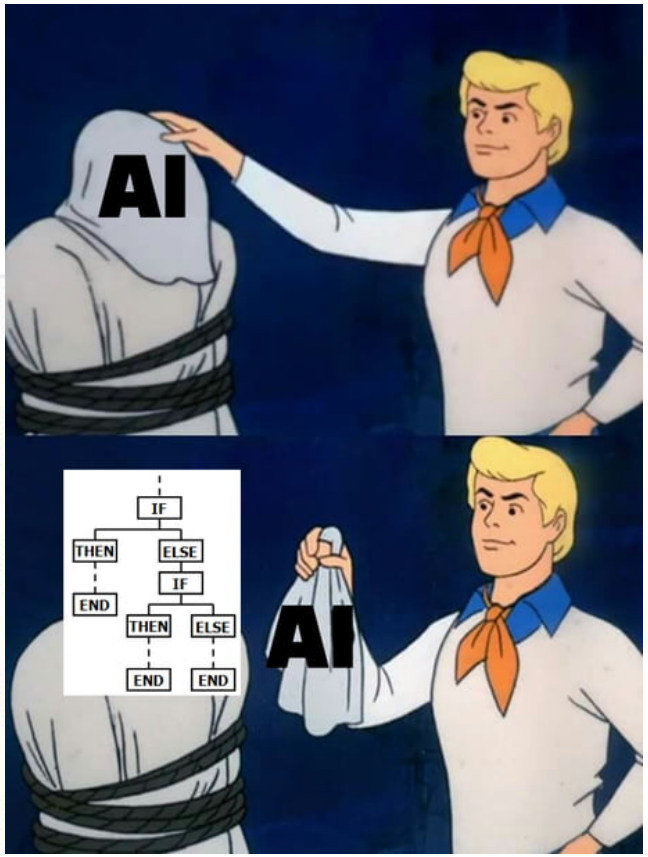
```
find_signature:
                mov     ecx, [ebp+4]
                xor     eax, eax

search_loop:                            ; CODE XREF: seg000:0000001C↓j
                cmp     dword ptr [eax+ecx], 0AABBCCDDh
                jz      short init
                inc     eax
                cmp     eax, 100h
                jb      short search_loop
                pop     ebp
                retn
; ---------------------------------------------------------------------------
```

```
$egg_hunter = {
8B 4D 04 31 C0 81 3C 08 ?? ?? ?? ?? 74 0A
40 3D 00 01 00 00 72 EF 5D C3
}
```

Traditional Methods for Mal ID

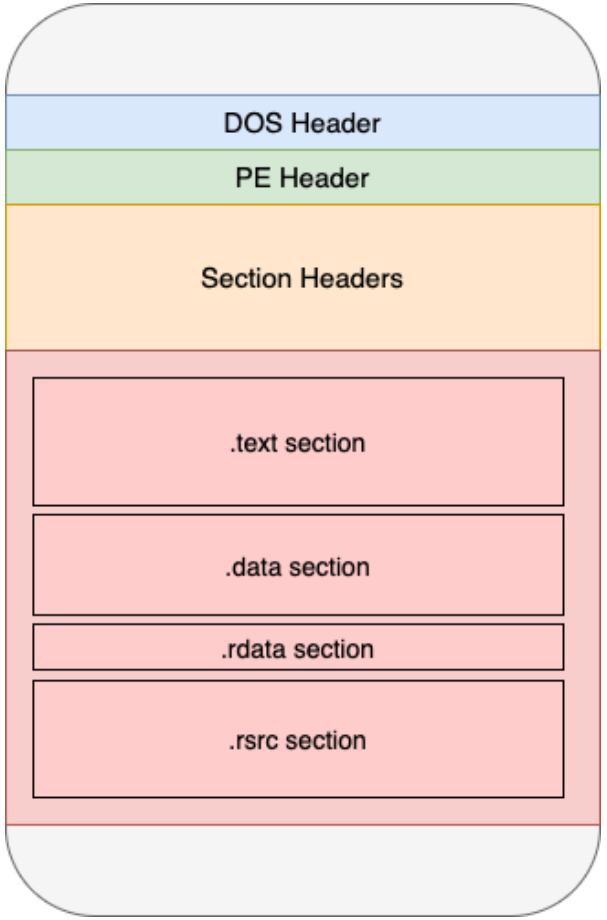
- AV Engine identification
  - Uses different approaches for detection
  - Signature, Heuristic Engine, ML Engine

Antivirus	Result
Ad-Aware	Trojan.GenericKD.5057860
AegisLab	Uds.Dangerousobject.Multi!c
AhnLab-V3	Trojan/Win32.WannaCryptor.R200571
Antiy-AVL	Trojan[Ransom]/Win32.Scatter
Arcabit	Trojan.Generic.D4D2D44
Avira (no cloud)	TR/AD.RansomHeur.aexdn
BitDefender	Trojan.GenericKD.5057860
Bkav	W32.RansomwareTBE.Trojan
CAT-QuickHeal	Ransom.WannaCrypt.A4



Short intro to PE file format

- File Headers
  - Import Table
  - Section Headers
- PE Sections
  - Text
  - Data
  - Rdata
  - Bss
  - Rsrc



Static Features

- Strings
- Section hashes
- ImpHash
- Fuzzy Hash (ssdeep)

Basic Properties

MD5	e1ac12483c2871ef2c5fbca7a450d562
SHA-1	db9491b178f81e7c228375bcc74bba792eef61e2
SHA-256	58d645eae34d49224d5b68cdc4d665a23383b34484be2ea3c946014f76770561
Vhash	035046551d551081z11zf1z41f5z4cz1ffz
Authentihash	f69b4cb34f8091c741dbe90854ed13f2cc7c2e2c7828b21183c676fb64c07b1c
Imphash	c8c95a8437da5797cf192c4cf27b186f
SSDEEP	3072:leYLrN2UApXRBZaKBEd/UslN0r//a0tziOOdltWdnDhNuo:4Y9peR+KBjziOOPtWdnDvZ

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5
.text	4096	283916	284160	5.16	5cd3942de5ea918bbe1dece4a4444b1f
.rdata	290816	325	512	3.18	1e99cd685208ea9344286df011997612
.data	294912	23284	23552	5.15	02c2d4f51b7952a63e6c2f8fbb2353f1
.rsrc	319488	15600	15872	3.48	953133eec5e0a24c5cebc925095cd96c

Section Hashes

- Malware are often reused with slight configuration changes
- Newer versions of a malware can be released
  - Which means codebase has alterations but its mostly the same
- Changes to the configuration may effect:
  - .rdata/.data
  - .rsrc
  - .text on rare occasions
- Changes to the code effect:
  - .text
  - .rdata if string constants are modified
  - .data if global variables are modified
  - .rsrc if resources are modified

ImpHash (import hash)

- The order of PE imports is dictated by their declaration order
- So changes to code may not have effect on IAT (Import Address Table)
- Taking hash of the IAT -> ImpHash

Basic Properties ⓘ

MD5	e1ac12483c2871ef2c5fbca7a450d562
SHA-1	db9491b178f81e7c228375bcc74bba792eef61e2
SHA-256	58d645eae34d49224d5b68cdc4d665a23383b34484be2ea3c946014f76770561
Vhash	035046551d551081z11zf1z41f5z4cz1ffz
Authentihash	f69b4cb34f8091c741dbe90854ed13f2cc7c2e2c7828b21183c676fb64c07b1c
Imphash	c8c95a8437da5797cf192c4cf27b186f
SSDEEP	3072:leYLrN2UApXRBZaKBEd/UslN0r//a0tziOOdltWdnDhNuo:4Y9peR+KBjziOOPtWdnDvZ

Fuzzy Hash (ssdeep)

- Basic idea is splitting data into blocks and hashing each block
- This way we are able to preserve similarity within the hash
- Context Triggered Piecewise Hashing (“CTPH”)

	sum of window	97	195	294	297	300	303	306	309	312	315	318	321	324	327	330	333	336	339	342	345	348	351	354	357	360	363	291	219
Context based	rolling hash	1	3	6	1	4	**7**	2	5	0	3	6	1	4	**7**	2	5	0	3	6	1	4	**7**	2	5	0	3	3	3
Original	content	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1
	ascii sum	597						852								916								579					
VUUD	b64 char	V						U								U								D					

Fuzzy Hash Reference: https://tinyurl.com/fuzzy-hashing

Static Features

- Benefits
  - Most are easy to extract
  - Very scalable
- Disadvantages
  - File format dependent
  - Packers impede automated static analysis
  - Very easy to deceive

Dynamic Features

- Network traffic
- Filesystem activity
- API calls
- Memory features
- Registry activity

Dynamic Features

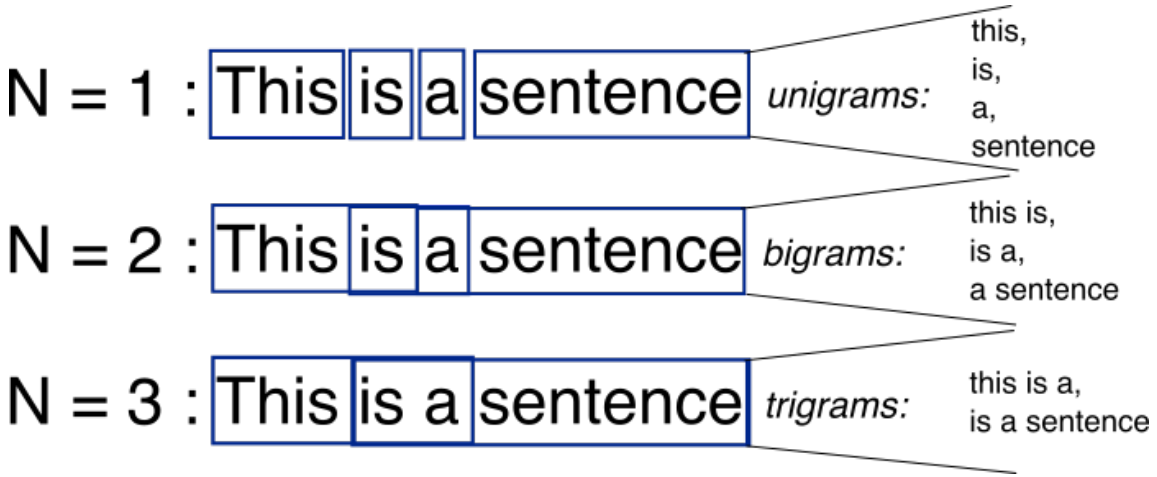
- Benefits
  - File format independent
  - Could beat the negative effect of packers
- Disadvantages
  - Extraction is very slow compared to static
  - Anti analysis greatly impede automated sandboxes
  - Therefore not scalable

Similarity Methods

- N-grams
- Bag of words
- Jaccard Index
- MinHash

N-gram Model

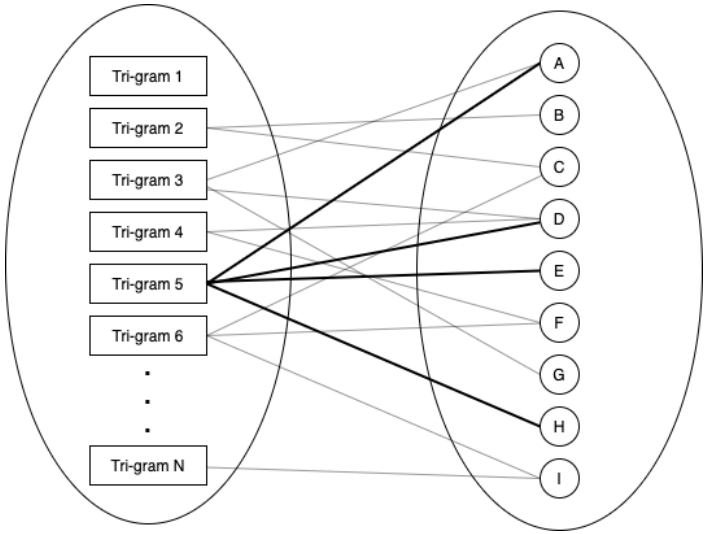
- Consept from Natural Language Processing
- Used for finding frequency of sequences
- Value of “N” is defined by the user

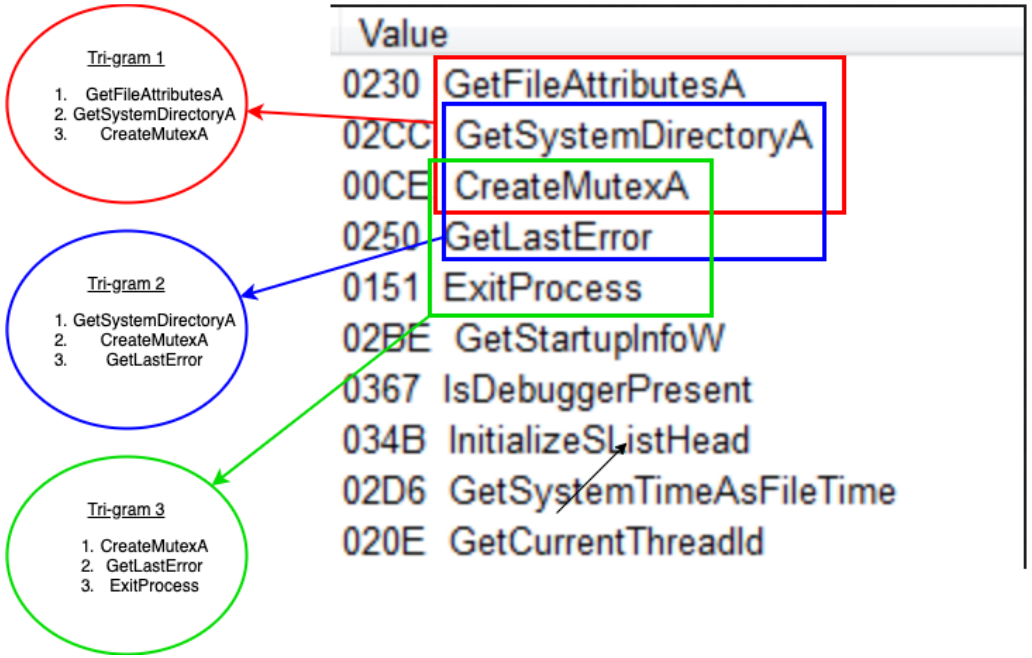


N-gram Reference: https://tinyurl.com/ngrams-1

N-gram on PE

- Instruction N-grams
- IAT N-grams
- Strings N-grams
- (Dynamic) API call N-grams





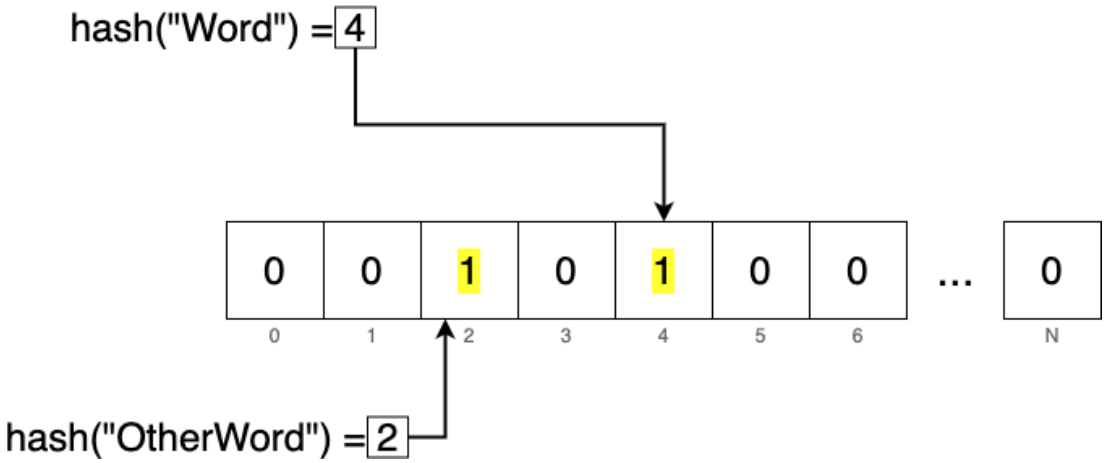
Bag of Words Vectorization

- If you have a known set of features you can use the BOW approach
- It’s a simple way to vectorize text

	Tri-gram 1	Tri-gram 2	Tri-gram 3	Tri-gram N
GetFileAttributesA	1	0	0	1
GetSystemDirectoryA	0	1	0	0
CreateMutexA	1	1	0	0
GetLastError	0	0	1	0
ExitProcess	0	0	0	1
GetStartupInfoW	0	1	0	0
IsDebuggerPresent	0	0	0	1
TerminateProcess	0	0	1	0
GetCurrentProcess	1	0	0	0
MessageBoxA	0	0	1	0

Bag of Words Issues

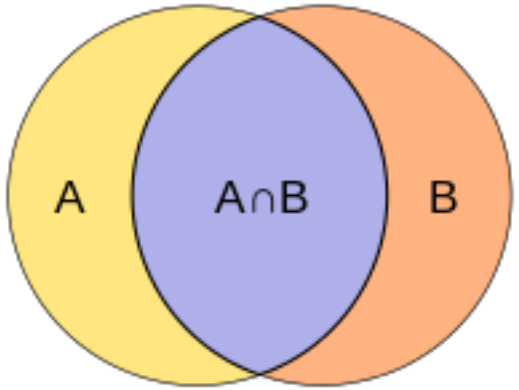
- Size of vocabulary grows as the size of dataset grows
- Searching keys in large dictionaries is slow; therefore encoding process will be gradually slower
- If you define a fixed size vocabulary, future samples may evade them
- Solution -> Hashing Trick!

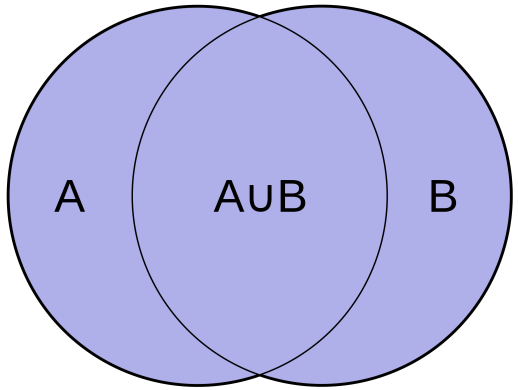


Jaccard Index

- It is a method to calculate similarity between two data

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





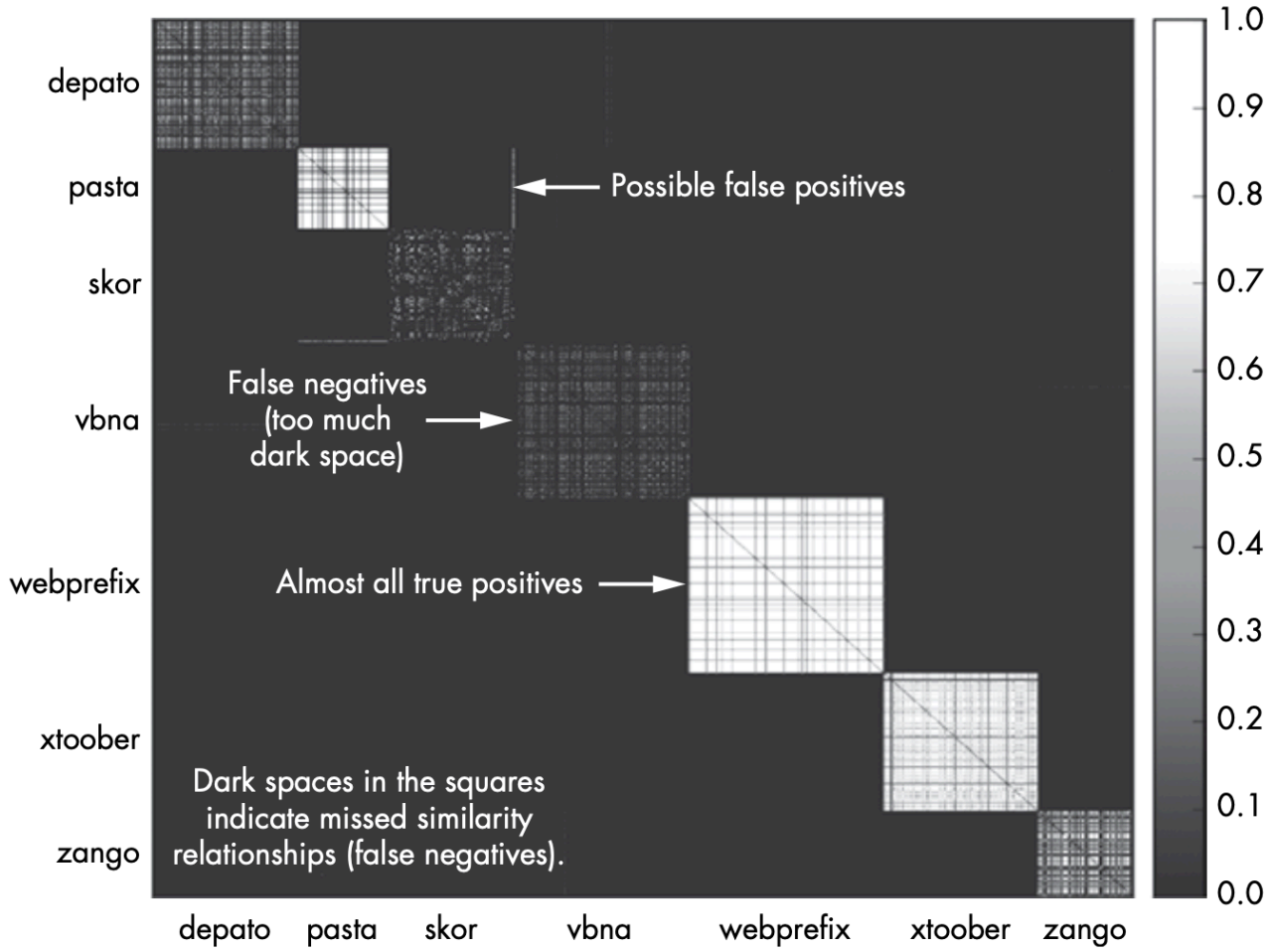
Similarity Matrix

- Jaccard similarity is a value between:
  - 0.0 and 1.0
- Calculate similarity for every sample in the dataset

	Sample 1	Sample 2	Sample 3	Sample 4
Sample 1	Similarity between 1 and 1	Similarity between 1 and 2	Similarity between 1 and 3	Similarity between 1 and 4
Sample 2	Similarity between 2 and 1	Similarity between 2 and 2	Similarity between 2 and 3	Similarity between 2 and 4
Sample 3	Similarity between 3 and 1	Similarity between 3 and 2	Similarity between 3 and 3	Similarity between 3 and 4
Sample 4	Similarity between 4 and 1	Similarity between 4 and 2	Similarity between 4 and 3	Similarity between 4 and 4

Reference: Malware Data Science

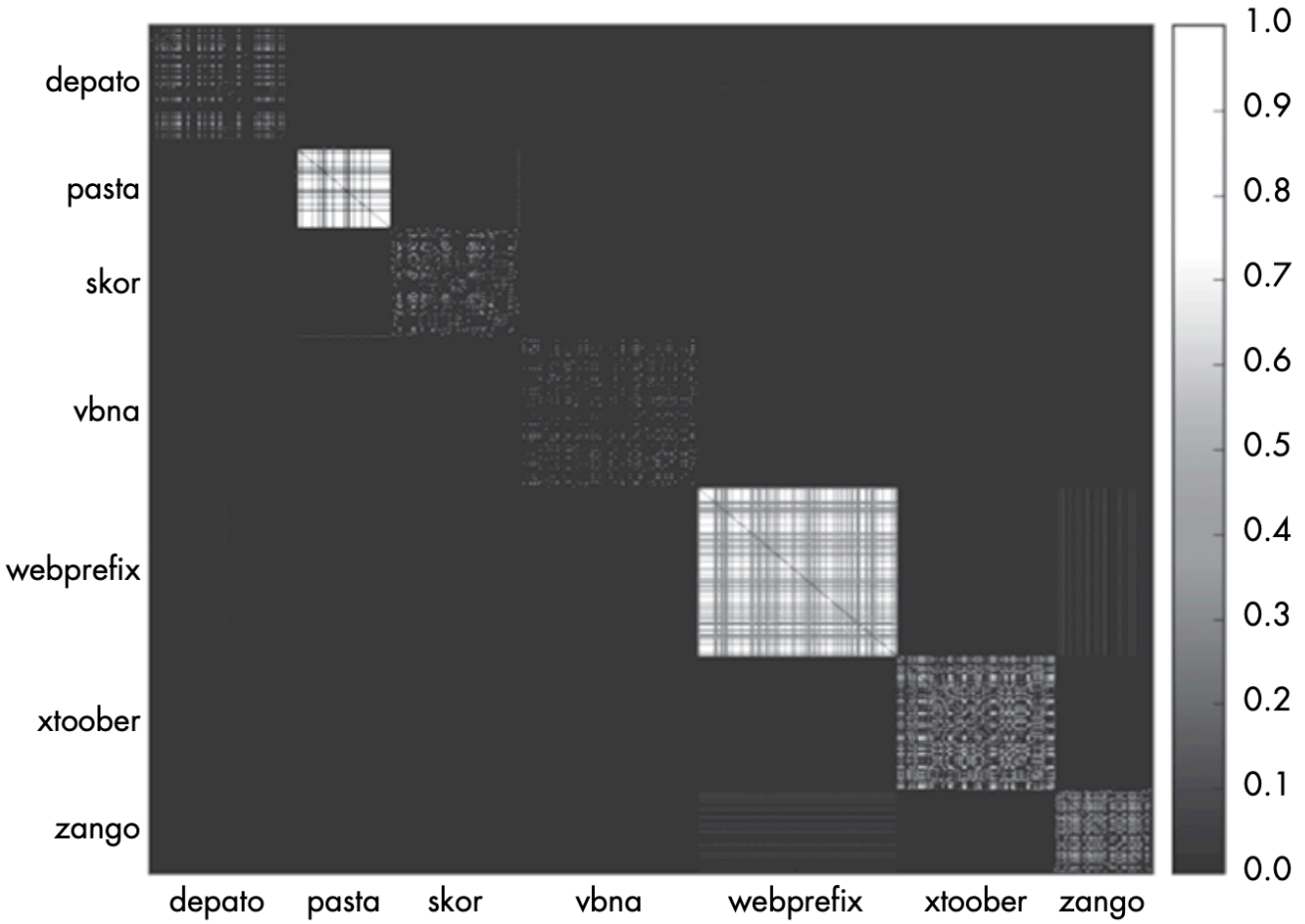
Example similarity



Instruction-seq based similarity

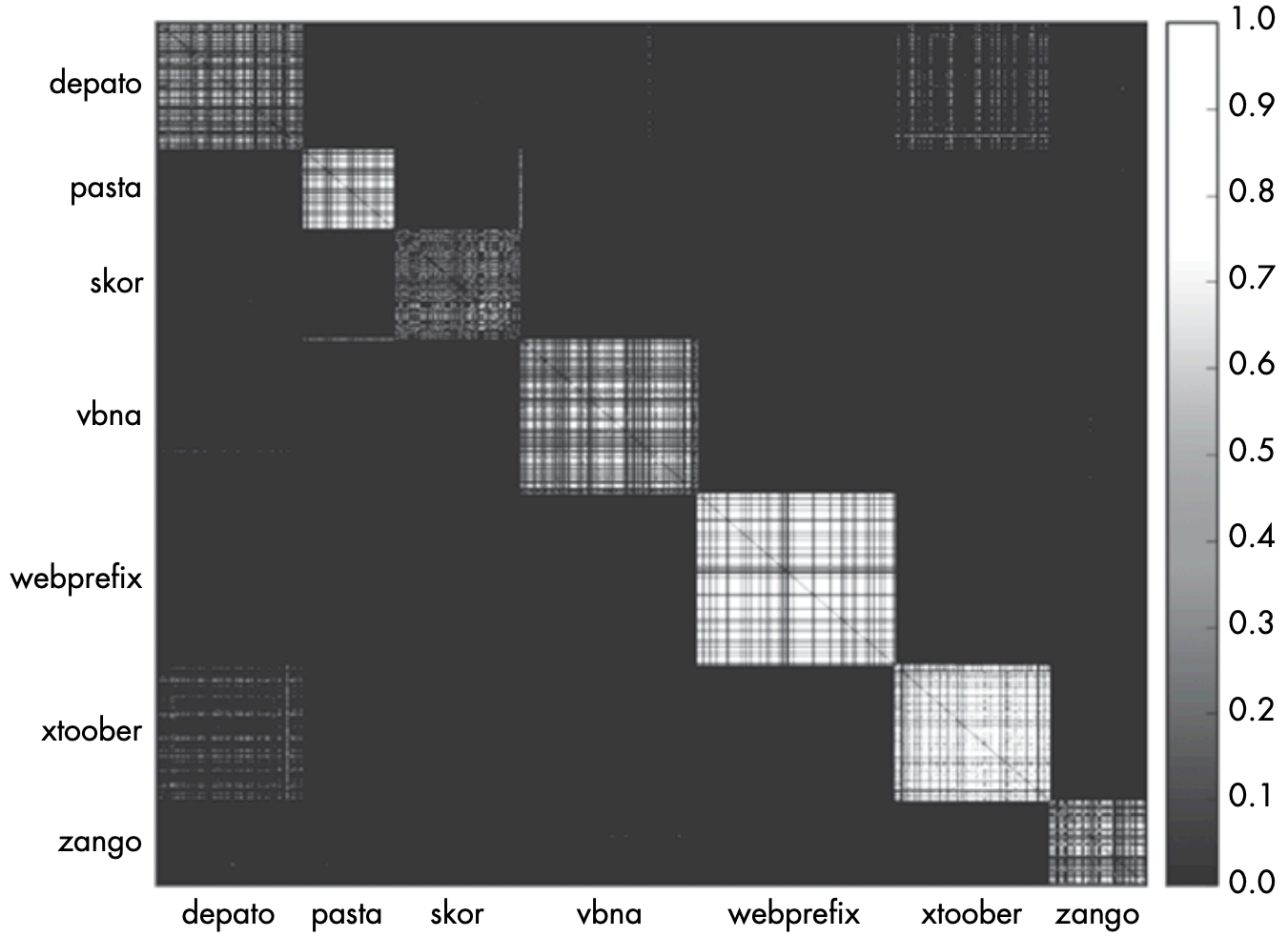
- Very poor results:
  - EXE Packers
  - Different compilers
  - Compiler settings
  - Optimization

WYSINWYX!!!



Strings based similarity

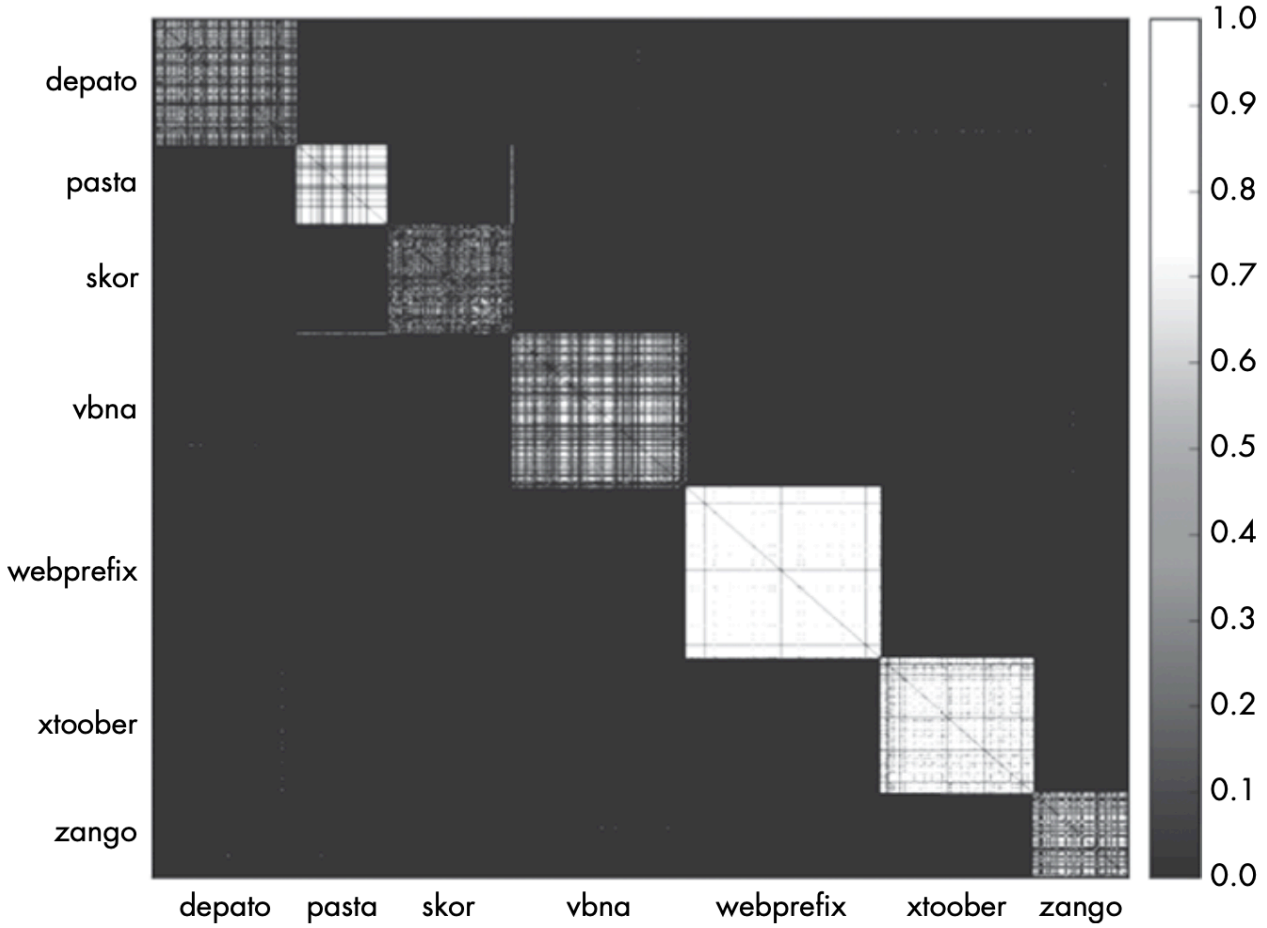
- Better than instruction sequence approach
- FP/FN rates are still high
- This is only seven families



Import Address Table based similarity

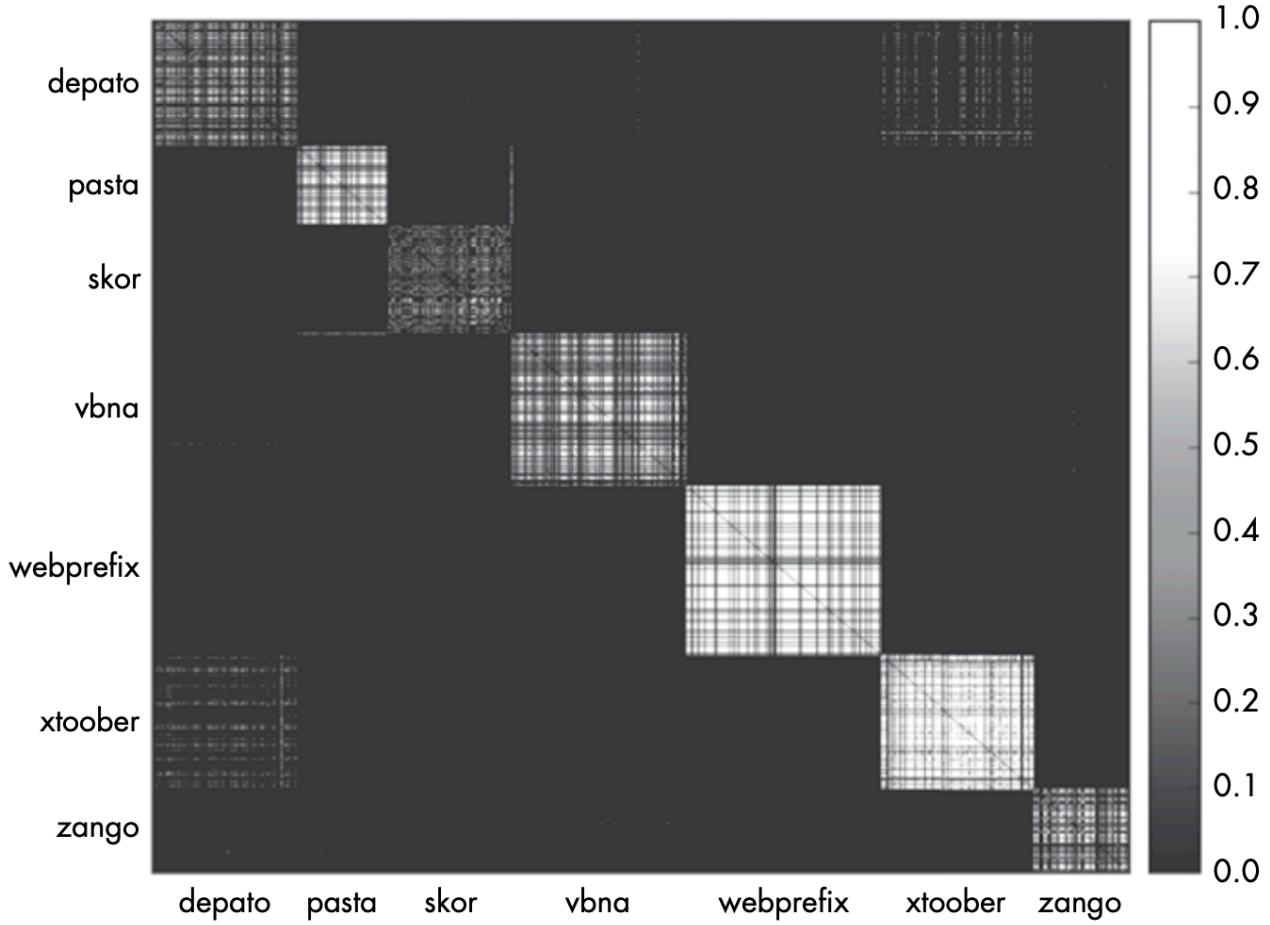
- Best results so far:
  - FN rate is still high
  - But FP rate is very low!!!

Remember the hypothesis regarding IAT from ImpHash



Dynamic API Call based similarity

- Dynamic approach is not so good either ☹
- Many problems facing automated dynamic analysis
  - Dependency problems
  - Anti-analysis features
  - Lack of user interaction



MinHash

- Jaccard calculation is very slow
- Therefore MinHash -> Locality-Sensitive Hashing (“LSH”)
- Select N random pieces from data then hash each one
- “Random Hash Function” (a.x + b % c)
- Comparing hashes give approximiate jaccard similary

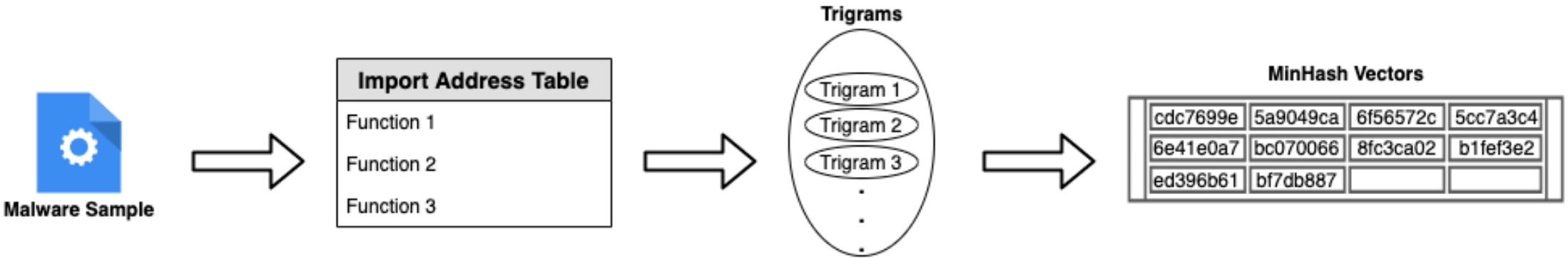
```
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python lsh.py
['Hello', 'World', 'Merhaba', 'Dunya']
['0x2ffb4c6e', '0x1450de6c', '0x61859bc', '0x61e2d311L', '0x12e3ab3e']
['World', 'Hello', 'Dunya', 'Merhaba']
['0x2ffb4c6e', '0x1450de6c', '0x61859bc', '0x61e2d311L', '0x12e3ab3e']
```

MinHash

- MinHash is a probabilistic method
- Therefore increasing the hash size also increase sensitivity
- To change a hash of N=10 you need to change ~%10 of the data
- To change a hash of N=20 you need to change atleast ~%5 of the data
- To change a hash of N=50 you need to change atleast ~%2 of the data

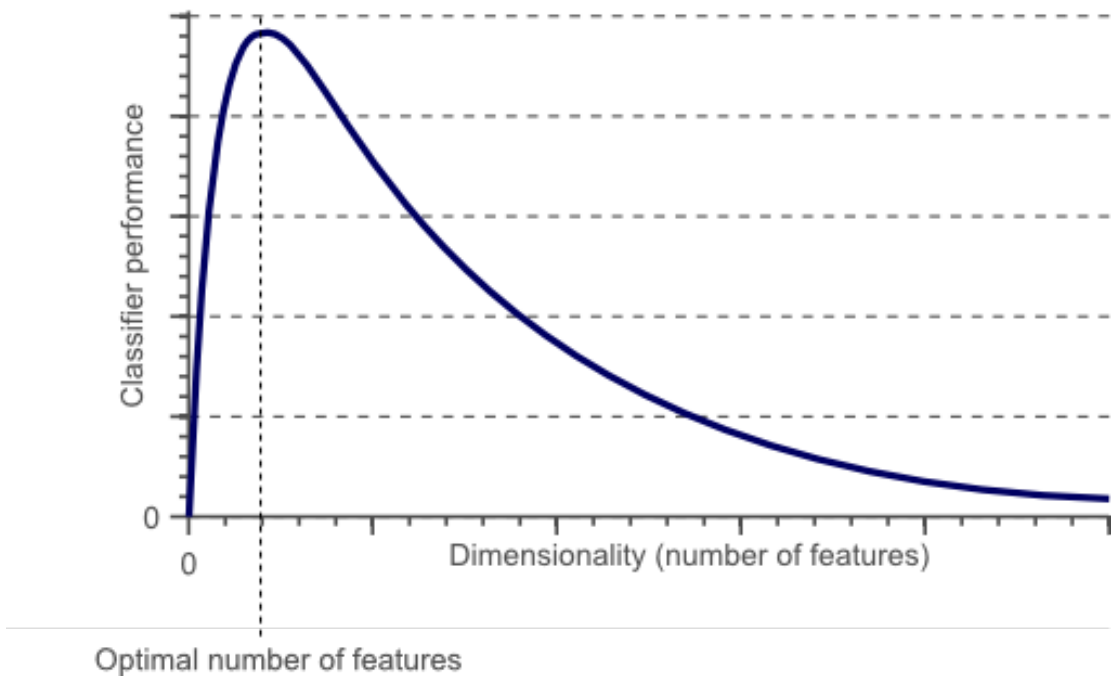
Our Method

- Imphash is insensitive to accompanying usage of functions
- Cryptographic hash function is used within Imphash
- Lets make an IAT hash that can tolerate slight changes and sensitive to accompanying usage of functions



MinHash Sensitivity vs. Curse of Dimensionality

- Longer MinHash (more sensitive) == More Dimensions
  = POOR PERFORMANCE!
- To avoid we will limit the N to 10



Ryuk Ransomware Samples

- Will compare our IAT Hash with other similarity hashing methods
- Lets test on two Ryuk samples to see how they will perform

SHA-256	2d3f396916cd6a84ec5cd89661a444650683f950a406036eec2882d555c95598
Vhash	015066655d1555155az57vza7z
Authentihash	e31d9cbd6b7ffb98ceb89d8cac612022f4c61b57832fa093f7f79a472b363129
Imphash	c58506a029779c7c484e16d07a842d9e
SSDEEP	3072:mPOAsyWPp4VHK3qgFbscZ8dS40uUJbFKtDfvZGi:vtPSK6gFAcOwAtDfv

SHA-256	7465a3de8afaacba99d8bf27d06b6e8702c2baae28b95b3a68749e45bd7e3030
Vhash	015046655d151098z5bhz2bza7z
Authentihash	95e7523ca08418e0798a9747deb3afbc76b5652b9832762e009ec30c794a8bc4
Imphash	b8618e50f04dad9a118a51e1abfe6e25
SSDEEP	3072:WphSHtkQDydbYLcXSz1IbYL+EGAQYqRGE8uRO:S4HL2xLqGbjEvIRO

Ryuk Ransomware Samples

- All of them (except Vhash) failed
- Our method shows 2/10 match!!

```
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python iat_hash.py
['0x11563e41', '0x1847c57', '0x42af5bc', '0x7bb581', '0x29db7e8', '0x3dfad3L', '0x61405a3', '0x58cee1b', '0xac480f2', '0x27e906']
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python iat_hash.py
['0x315136c', '0x8dd192', '0x21c6a21', '0x7bb581', '0x1f7549f', '0x511a06bL', '0x19d2619', '0x55f2d76', '0x1901915', '0x27e906']
```

Ryuk Ransomware Samples

- If we define N=2 match rate becomes 3/10

```
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python iat_hash.py
['0x5a83b1f', '0x353f757', '0x18951ff', '0x1d1870', '0x8eb1037', '0xd3f2dfa', '0x60afa5c', '0x78655e', '0x335ab9b', '0x31eb6d5']
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python iat_hash.py
['0x5a83b1f', '0x4885c8f', '0xac9265', '0x1d1870', '0x42667e2', '0x255139a', '0x503a6a6', '0x78655e', '0x9c63ae', '0x1558b58']
```

Netwalker Ransomware Samples

- When N=3 result is 0/10
- When N=2 result is 2/10

```
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python iat_hash.py
['0x68b2f6', '0x7e9902', '0xac9265', '0x1319c60', '0x1e923b', '0x15cc705L', '0x581bc4', '0xc1a852', '0x4e76efe', '0x3db3751']
(machinelearning) Robins-MacBook-Pro:machinelearning robindimyan$ python iat_hash.py
['0x1693d7a', '0x1e63ade', '0xac9265', '0x16222f2', '0x4ecc4ed', '0x6ee596', '0x255d4f7', '0x5961d5', '0x517cc2', '0x51fa417']
```

Questions?