COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Detecting Organized e-Commerce Fraud

Dr. Martin “Doc” Carlisle

eCommerce Fraud

- Obtain physical goods through deception
  - Amazon, e-Bay, etc.
- 3-5% of online orders, $50B revenue
- Want to cancel bad orders, but not good ones!
  - Authors say this is $100B in losses

Organized eCommerce Fraud

- Limited time, several orders to limited geographical area
- Zalando (largest online apparel retailer in Europe) lost €18.5M in one quarter in 2015 to this

Method

- Unsupervised categorical clustering
  - “recursive agglomerative clustering”
    - Specifically designed for this problem
    - Creates small clusters
    - Samples to process large # of orders in less time
  - Two outputs
    - A1 – orders to screen
    - A2 – orders to automatically cancel

Fraud Detection Pipeline

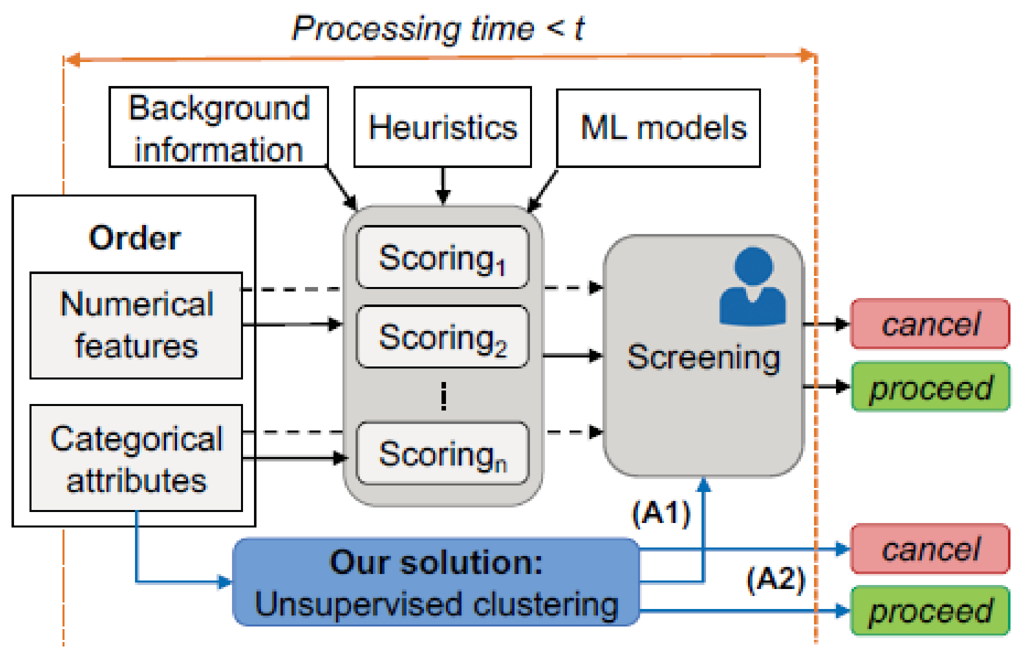


Figure 1: Fraud detection pipeline. Final cancellation is decided by human analysts during screening. Our solution supports screening (A1) and automatically cancel frauds (A2).

Claimed Benefits

- Expect frauds to be clustered more than legitimate orders
- Several orders are easier to analyze than isolated ones

Order attributes

- Customer Acust (9 attributes): related to the electronic identity of the customer, e.g., email address, IP address, etc.
- Delivery Adel (3 attributes): related to the means used for order delivery, e.g, pickup point, delivery type, etc.
- Shipping Aship (7 attributes): related to identity and location (address) of the person receiving the order.
- Payment Apay (11 attributes): related to payment method, e.g., bank transfer, credit card suffix, etc.
- Billing Abill (7 attributes): related to identity and location (address) of the person paying the order.

Difficulties

- Imbalanced attribute cardinality
  - Prevents numerical encoding of attributes
- Imbalanced classes
  - Fraud to legit is about 1:50
- No ground truth for campaigns
- Scale of data
  - 100,000s of orders per day– hard for clustering algs

Requirements

- Generate small clusters
  - Fraud campaign is 10s-100s of orders
- Minimize cluster impurity
  - Don’t combine legit and fraud
- Maximize clustered fraud
  - Singleton frauds aren’t caught by this method
- Min execution time
  - Few hours tops

Agglomerative Clustering

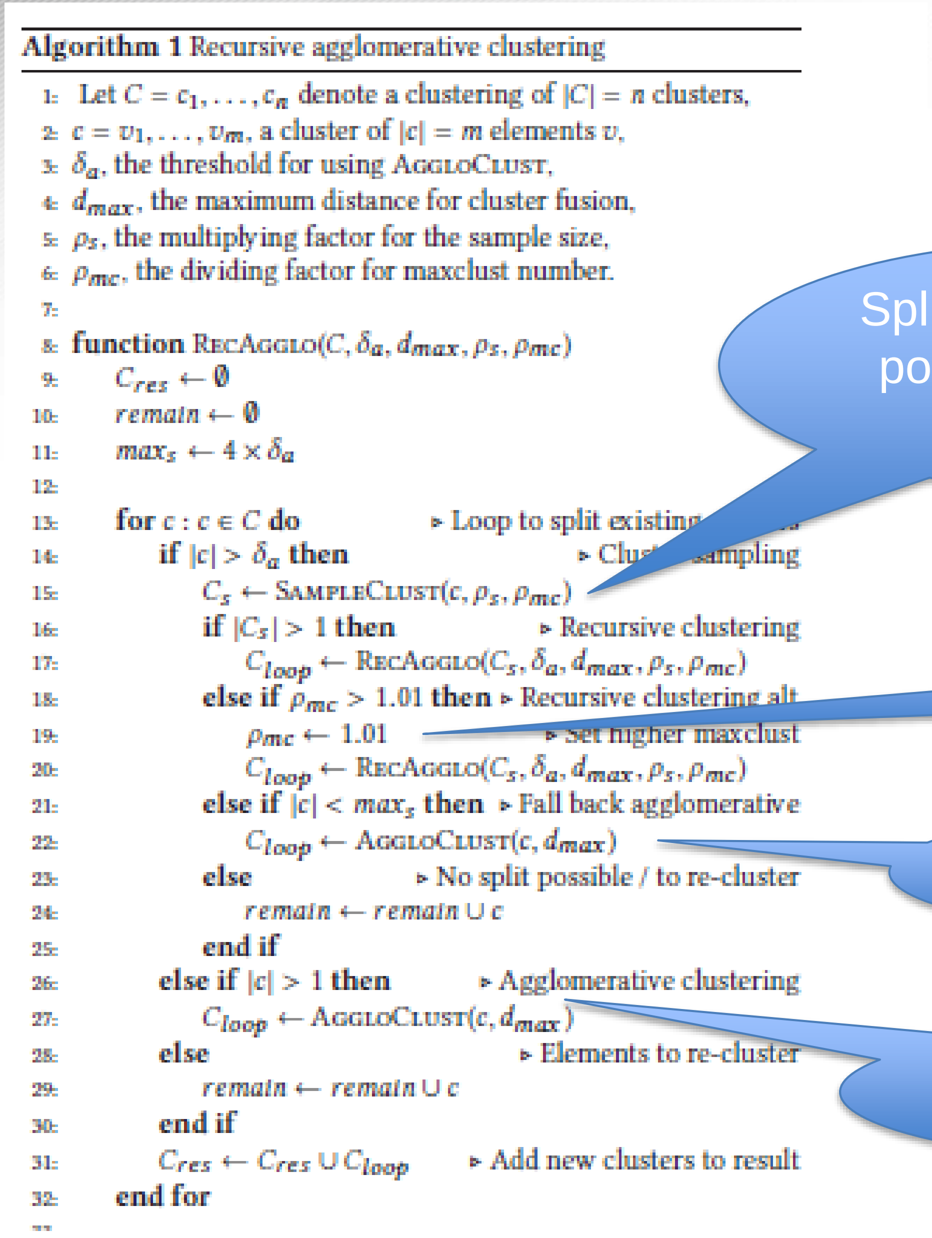
- Bottom-up approach
- Place each order in singleton cluster
- Combine clusters with smallest distance until all are linked
  - Distance between clusters is via “linkage”
    - Single linkage is min distance between any two points in each cluster
    - O(n^2) – doesn’t scale well

Adding sampling

- Random sample of reference elements clustered
- Remaining ones are assigned to initially formed clusters
  - Distance computations not n x n, but |samples| x n
  - E.g. number of samples is O(log(n))

Recursive Algorithm

- Split large clusters into smaller ones with SampleClust
- For “small enough” do AggloClust



Algorithm 1 Recursive agglomerative clustering

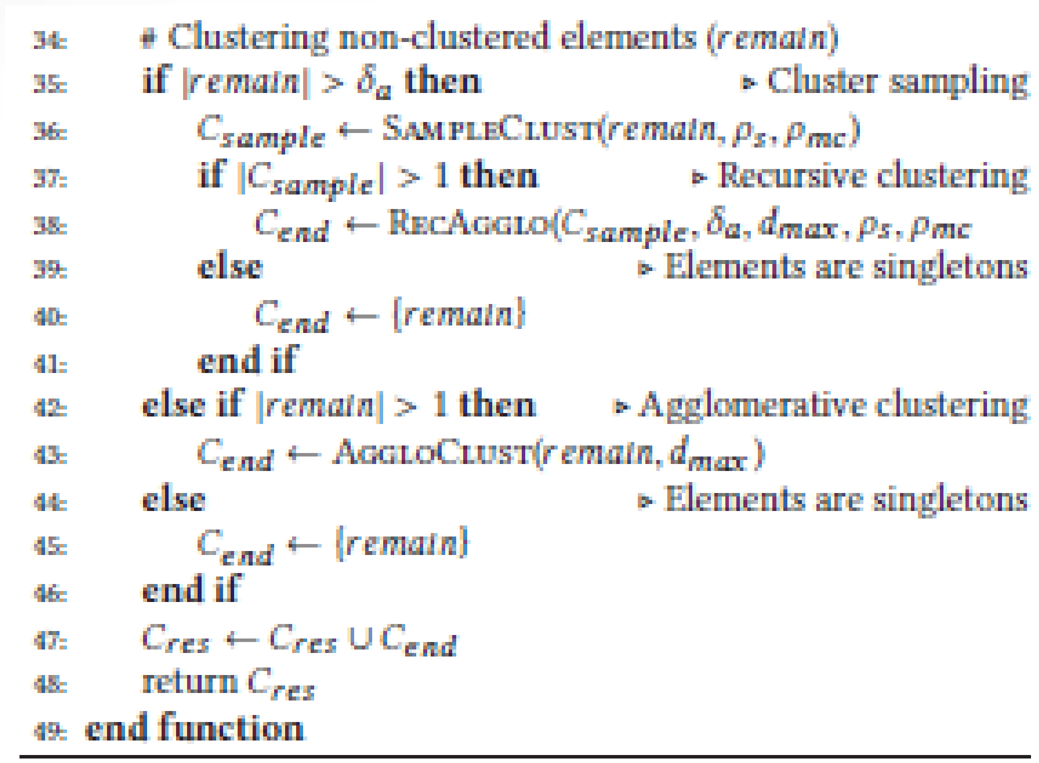
```
Let $C = c_1, \ldots, c_n$ denote a clustering of $|C| = n$ clusters,
$c = v_1, \ldots, v_m$, a cluster of $|c| = m$ elements $v$,
$\delta_a$, the threshold for using AggloClust,
$d_{max}$, the maximum distance for cluster fusion,
$\rho_s$, the multiplying factor for the sample size,
$\rho_{mc}$, the dividing factor for maxclust number.

function RecAgglo($C, \delta_a, d_{max}, \rho_s, \rho_{mc}$)
    $C_{res} \leftarrow \emptyset$
    $remain \leftarrow \emptyset$
    $max_s \leftarrow 4 \times \delta_a$

    for $c : c \in C$ do                          ▷ Loop to split existing
        if $|c| > \delta_a$ then                  ▷ Clu      ampling
            $C_s \leftarrow$ SampleClust($c, \rho_s, \rho_{mc}$)
            if $|C_s| > 1$ then                   ▷ Recursive clustering
                $C_{loop} \leftarrow$ RecAgglo($C_s, \delta_a, d_{max}, \rho_s, \rho_{mc}$)
            else if $\rho_{mc} > 1.01$ then       ▷ Recursive clustering alt
                $\rho_{mc} \leftarrow 1.01$       ▷ Set higher maxclust
                $C_{loop} \leftarrow$ RecAgglo($C_s, \delta_a, d_{max}, \rho_s, \rho_{mc}$)
            else if $|c| < max_s$ then            ▷ Fall back agglomerative
                $C_{loop} \leftarrow$ AggloClust($c, d_{max}$)
            else                                  ▷ No split possible / to re-cluster
                $remain \leftarrow remain \cup c$
            end if
        else if $|c| > 1$ then                    ▷ Agglomerative clustering
            $C_{loop} \leftarrow$ AggloClust($c, d_{max}$)
        else                                      ▷ Elements to re-cluster
            $remain \leftarrow remain \cup c$
        end if
        $C_{res} \leftarrow C_{res} \cup C_{loop}$   ▷ Add new clusters to result
    end for
```

Split large cluster into possibly many, then do Recursion

It didn’t split, maybe try harder

It didn’t split, but not too big to brute force

Agglomerative for smaller ones



```
    # Clustering non-clustered elements ($remain$)
    if $|remain| > \delta_a$ then                              ▷ Cluster sampling
        $C_{sample} \leftarrow$ SampleClust($remain, \rho_s, \rho_{mc}$)
        if $|C_{sample}| > 1$ then                        ▷ Recursive clustering
            $C_{end} \leftarrow$ RecAgglo($C_{sample}, \delta_a, d_{max}, \rho_s, \rho_{mc}$
        else                                          ▷ Elements are singletons
            $C_{end} \leftarrow \{remain\}$
        end if
    else if $|remain| > 1$ then                   ▷ Agglomerative clustering
        $C_{end} \leftarrow$ AggloClust($remain, d_{max}$)
    else                                              ▷ Elements are singletons
        $C_{end} \leftarrow \{remain\}$
    end if
    $C_{res} \leftarrow C_{res} \cup C_{end}$
    return $C_{res}$
end function
```

Running time

- All AgloClusts are constant time (fixed size max)
- SampleClust is O(nlog(n)) and this might happen n times recursively!
  - O((nlog(n))^n) – yuck!
  - But this doesn’t really happen

Computation Time

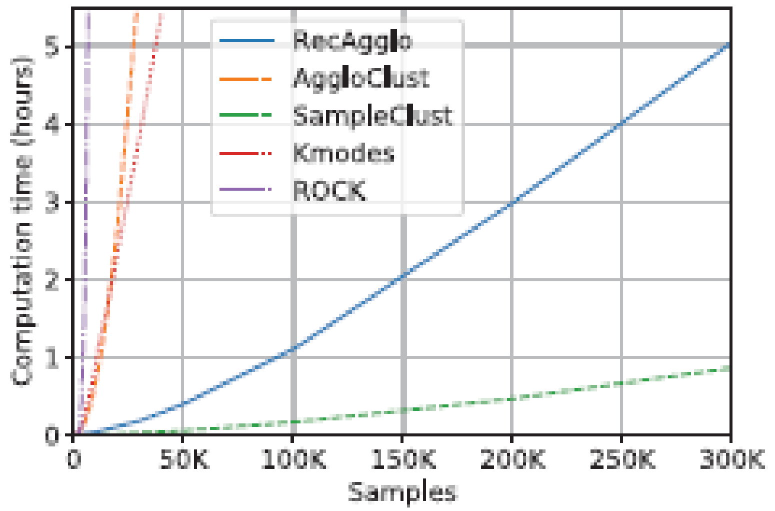


Figure 5: Computation time (averaged over 5 runs) vs. sample size for 5 clustering algorithms. Only SampleClust and RecAgglo can scale to large datasets.

Clustering results (15000)

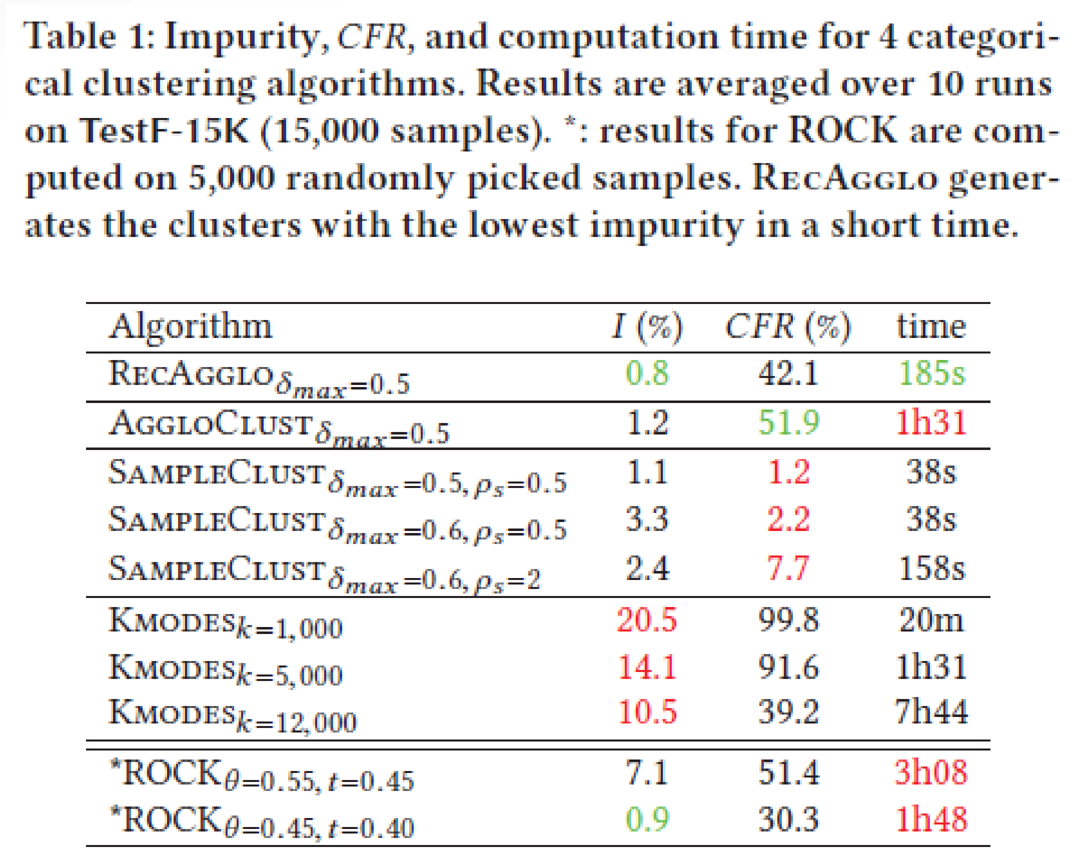


Table 1: Impurity, *CFR*, and computation time for 4 categorical clustering algorithms. Results are averaged over 10 runs on TestF-15K (15,000 samples). *: results for ROCK are computed on 5,000 randomly picked samples. RecAgglo generates the clusters with the lowest impurity in a short time.

Algorithm	I (%)	CFR (%)	time
RecAgglo$_{\delta_{max}=0.5}$	0.8	42.1	185s
AggloClust$_{\delta_{max}=0.5}$	1.2	51.9	1h31
SampleClust$_{\delta_{max}=0.5,\rho_s=0.5}$	1.1	1.2	38s
SampleClust$_{\delta_{max}=0.6,\rho_s=0.5}$	3.3	2.2	38s
SampleClust$_{\delta_{max}=0.6,\rho_s=2}$	2.4	7.7	158s
Kmodes$_{k=1,000}$	20.5	99.8	20m
Kmodes$_{k=5,000}$	14.1	91.6	1h31
Kmodes$_{k=12,000}$	10.5	39.2	7h44
*ROCK$_{\theta=0.55,t=0.45}$	7.1	51.4	3h08
*ROCK$_{\theta=0.45,t=0.40}$	0.9	30.3	1h48

How far apart are they?

- Hamming weight computes weighted sum of how many attributes differ

$$Hamming(u, v) = \frac{1}{d}\sum_{l=1}^{d} w_l \times (u_l \neq v_l) \qquad (1)$$

- They weight by number of possible values of each attribute with sigmoid and max difference of 3x

$$w_l^s = 1 + 2 \times \left(1 - \frac{R_l^{-1}}{median(R_l^{-1}) + R_l^{-1}}\right), w_l^s \in [1, 3] \qquad (2)$$

Results (I)

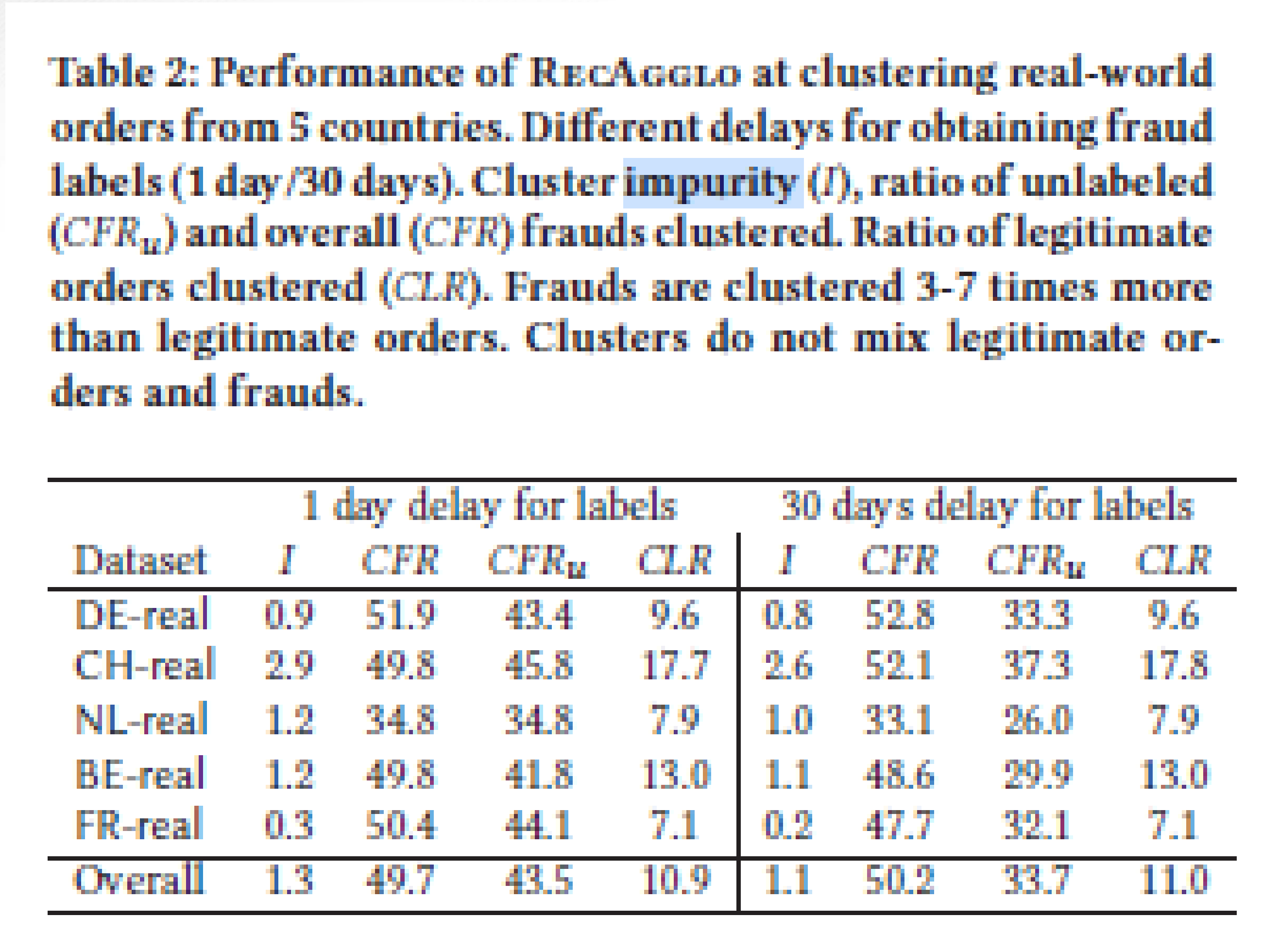


Table 2: Performance of RecAgglo at clustering real-world orders from 5 countries. Different delays for obtaining fraud labels (1 day/30 days). Cluster impurity (I), ratio of unlabeled (CFR_u) and overall (CFR) frauds clustered. Ratio of legitimate orders clustered (CLR). Frauds are clustered 3-7 times more than legitimate orders. Clusters do not mix legitimate orders and frauds.

	1 day delay for labels				30 days delay for labels			
Dataset	I	CFR	CFR_u	CLR	I	CFR	CFR_u	CLR
DE-real	0.9	51.9	43.4	9.6	0.8	52.8	33.3	9.6
CH-real	2.9	49.8	45.8	17.7	2.6	52.1	37.3	17.8
NL-real	1.2	34.8	34.8	7.9	1.0	33.1	26.0	7.9
BE-real	1.2	49.8	41.8	13.0	1.1	48.6	29.9	13.0
FR-real	0.3	50.4	44.1	7.1	0.2	47.7	32.1	7.1
Overall	1.3	49.7	43.5	10.9	1.1	50.2	33.7	11.0

Results (II)

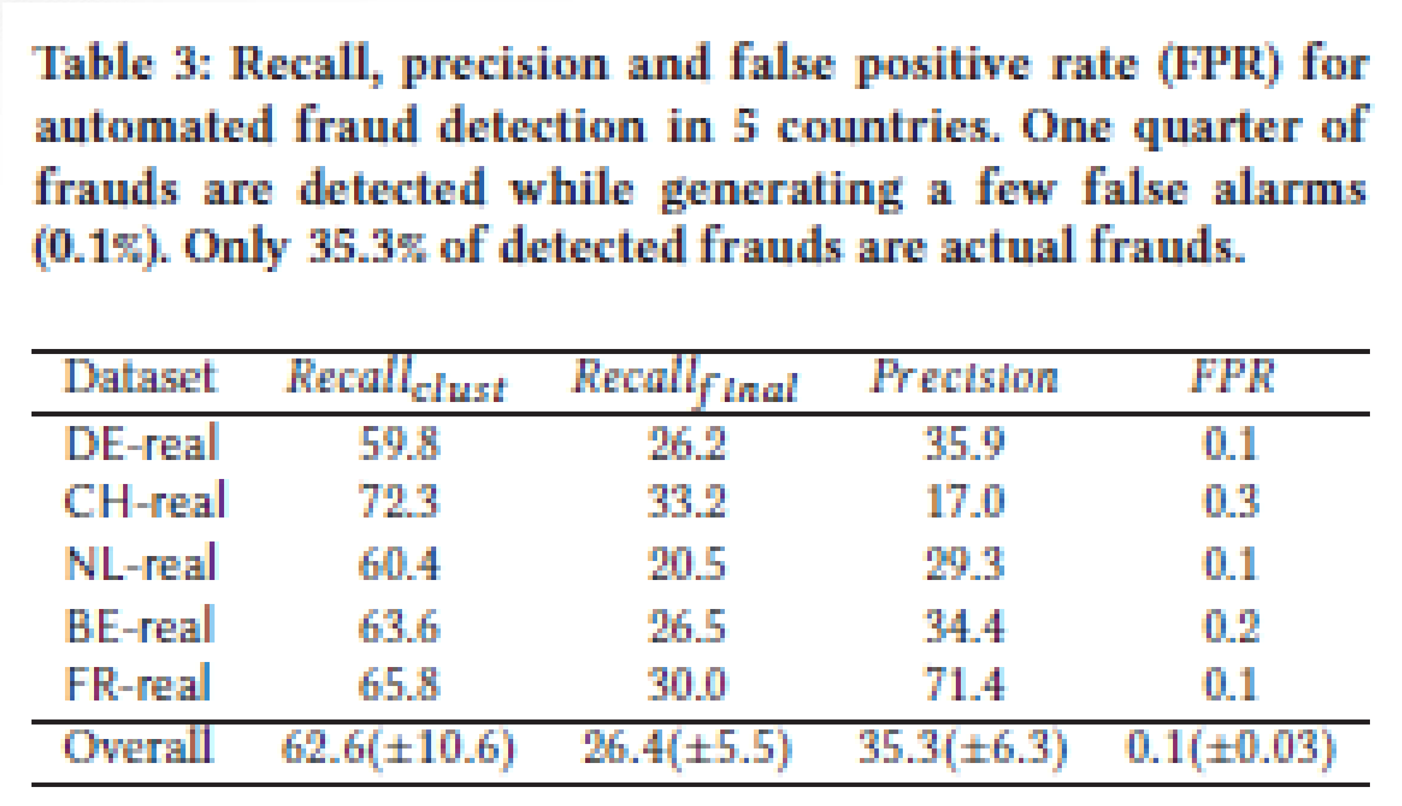


Table 3: Recall, precision and false positive rate (FPR) for automated fraud detection in 5 countries. One quarter of frauds are detected while generating a few false alarms (0.1%). Only 35.3% of detected frauds are actual frauds.

Dataset	$Recall_{clust}$	$Recall_{final}$	$Precision$	FPR
DE-real	59.8	26.2	35.9	0.1
CH-real	72.3	33.2	17.0	0.3
NL-real	60.4	20.5	29.3	0.1
BE-real	63.6	26.5	34.4	0.2
FR-real	65.8	30.0	71.4	0.1
Overall	62.6(±10.6)	26.4(±5.5)	35.3(±6.3)	0.1(±0.03)

What went wrong

We investigated further the characteristics of legitimate orders that our *label propagation* technique incorrectly identified as fraud. We computed the ratio of these orders that belong to four legitimate categories namely, (1) *fully* and (2) *partially returned* to the retailer (where a customer does not pay for all items and returns some of them), (3) *partly unpaid* (where items in the order remain unpaid while delivered) and (4) *canceled* by the customer. We observed that 94.7% of the false positives that degrade the Precision of *label propagation* belong to one of these four categories. The majority of the false positives (63.9%) are returned orders while 24.3% are partly unpaid orders.

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Survey of Data Mining/ML Cyber Intrusion Detection

Dr. Martin "Doc" Carlisle

ML/DM approaches (review)

- Unsupervised
  - Find patterns in unlabeled data
- Semi-supervised
  - Portion of data labeled
- Supervised
  - All data labeled

Data Mining/ML Process

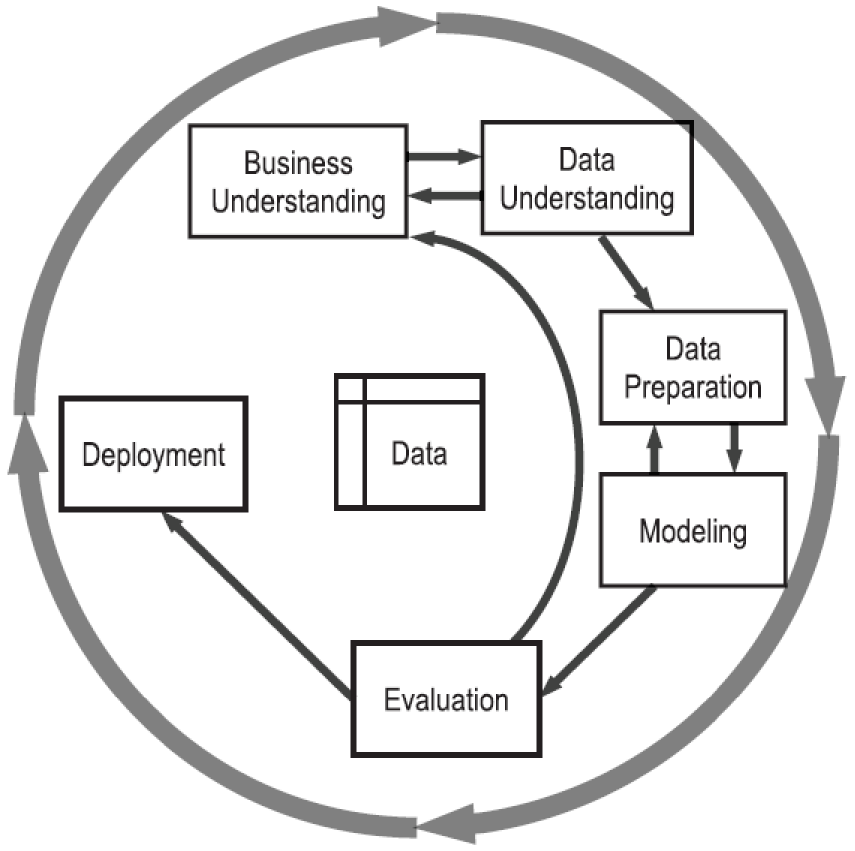


Fig. 1. CRISP-DM Process Diagram.

Metrics (review)

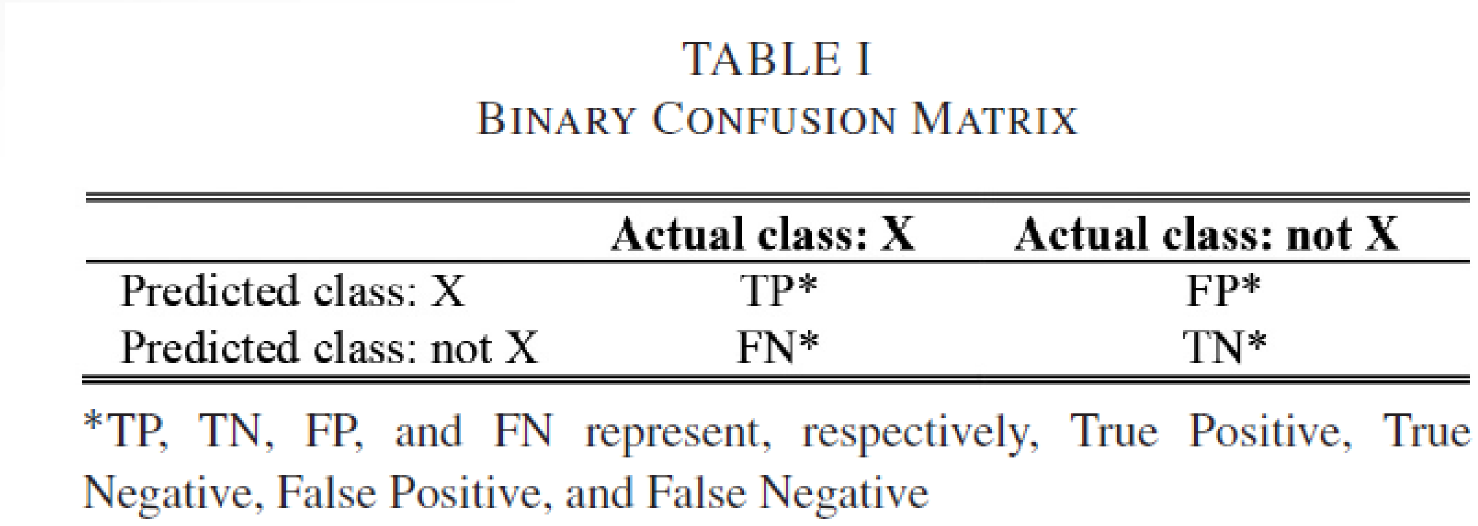


TABLE I
BINARY CONFUSION MATRIX

	Actual class: X	Actual class: not X
Predicted class: X	TP*	FP*
Predicted class: not X	FN*	TN*

*TP, TN, FP, and FN represent, respectively, True Positive, True Negative, False Positive, and False Negative

Metrics (review)

- Accuracy: (TP+TN)/(TP+TN+FP+FN)
  - What overall percent did you get right?
  - When classes unbalanced you can get high accuracy by always choosing 1 class
- Precision or Positive Predictive Value (PPV) : TP/(TP+FP)
  - What percent of my positives are real?
- Recall/Sensitivity/True Positive Rate/Detection Rate: TP/(TP+FN)
  - How percent do I catch?

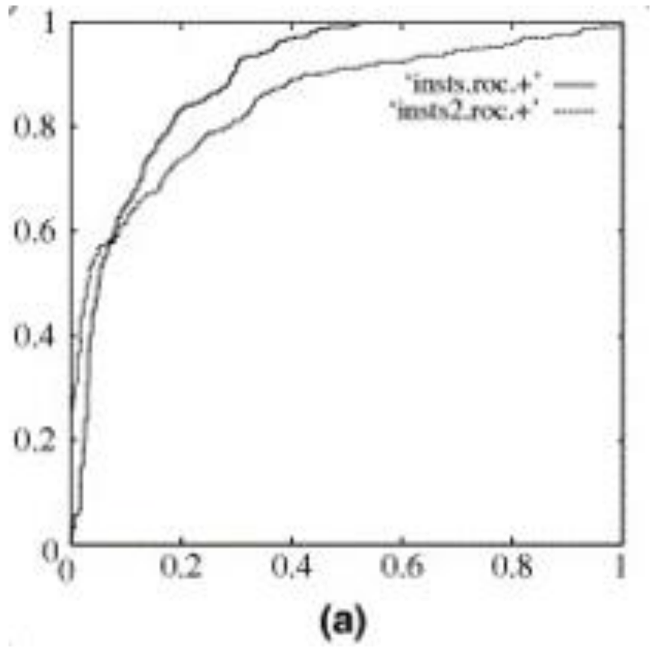
Metrics (review)

- Negative Predictive Value(NPV): TN/(TN+FN)
  - What percent of negative classifications are correct?
- Specificity or TN Rate: TN/(TN+FP)
  - What percent of negatives are marked correctly?
- FP Rate: FP/(TN+FP)
  - What percent are false positives?

COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

Metric Tradeoff (Review)

- ROC captures tradeoff between false positives and recall
  - – FP on x-axis, recall/sensitivity on y-axis



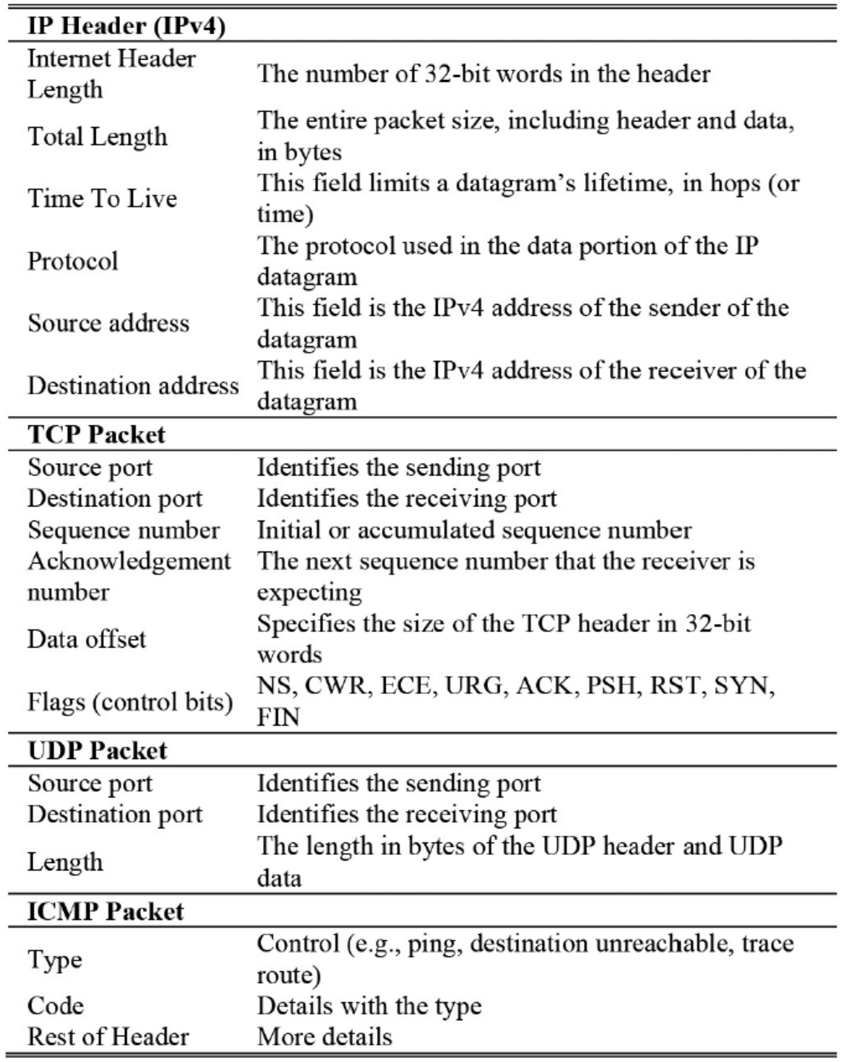
(a)

Multi-class metrics

- Overall accuracy
- Class detection rate
- Class FP rate

Intrusion Detection Data

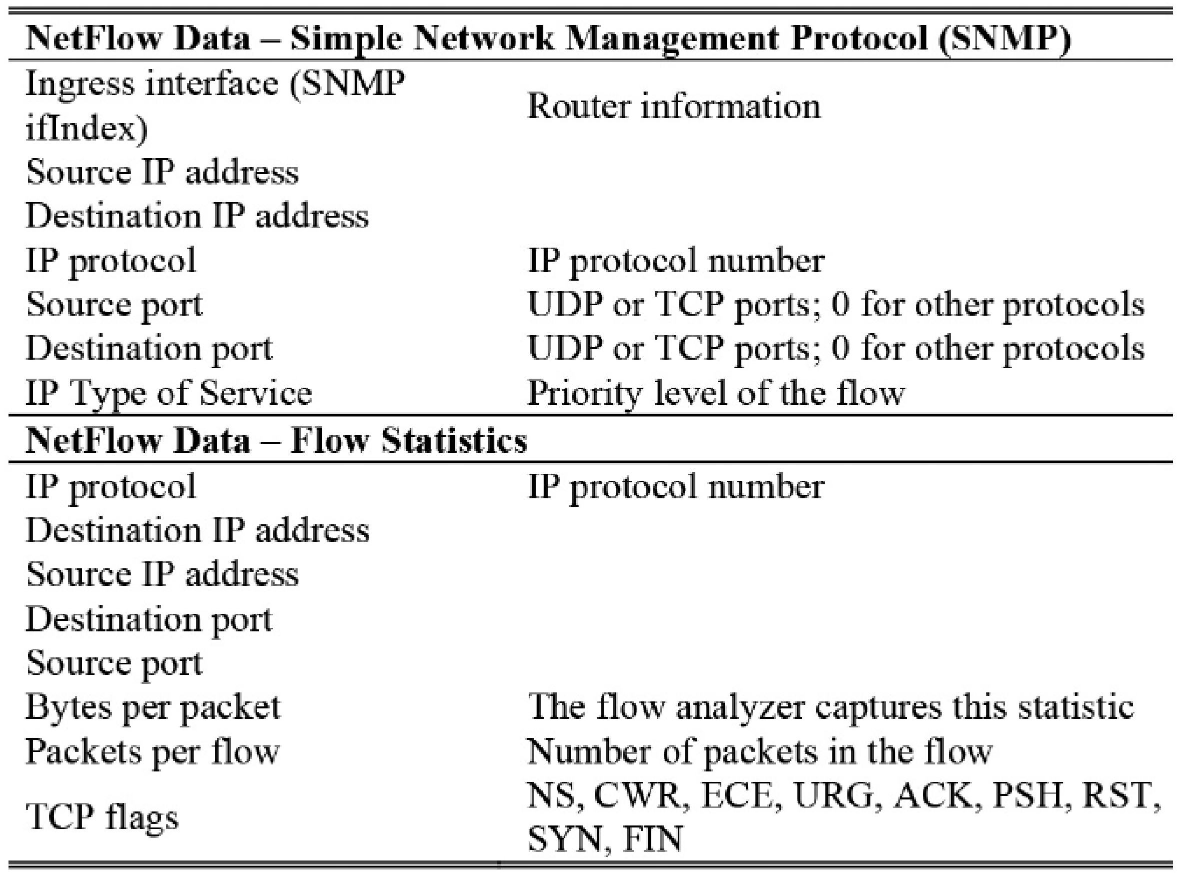
- Packet Level Data
  - TCPDUMP
  - Wireshark
  - …



IP Header (IPv4)	
Internet Header Length	The number of 32-bit words in the header
Total Length	The entire packet size, including header and data, in bytes
Time To Live	This field limits a datagram's lifetime, in hops (or time)
Protocol	The protocol used in the data portion of the IP datagram
Source address	This field is the IPv4 address of the sender of the datagram
Destination address	This field is the IPv4 address of the receiver of the datagram
TCP Packet	
Source port	Identifies the sending port
Destination port	Identifies the receiving port
Sequence number	Initial or accumulated sequence number
Acknowledgement number	The next sequence number that the receiver is expecting
Data offset	Specifies the size of the TCP header in 32-bit words
Flags (control bits)	NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN
UDP Packet	
Source port	Identifies the sending port
Destination port	Identifies the receiving port
Length	The length in bytes of the UDP header and UDP data
ICMP Packet	
Type	Control (e.g., ping, destination unreachable, trace route)
Code	Details with the type
Rest of Header	More details

Intrusion Detection Data

- NetFlow Data
  - Orig Cisco
  - …



NetFlow Data – Simple Network Management Protocol (SNMP)	
Ingress interface (SNMP ifIndex)	Router information
Source IP address	
Destination IP address	
IP protocol	IP protocol number
Source port	UDP or TCP ports; 0 for other protocols
Destination port	UDP or TCP ports; 0 for other protocols
IP Type of Service	Priority level of the flow
NetFlow Data – Flow Statistics	
IP protocol	IP protocol number
Destination IP address	
Source IP address	
Destination port	
Source port	
Bytes per packet	The flow analyzer captures this statistic
Packets per flow	Number of packets in the flow
TCP flags	NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN

Public Data Sets

- DARPA 1998, 1999
  - Note this paper in 2016Q2!
  - 1998: 9 weeks of data with attack simulations
    - 7 weeks training, 2 weeks testing
    - DoS, User to Root (U2R), Remote to Local (R2L), Probe/Scan
  - 1999: 5 weeks
    - 3 weeks training, 2 testing
    - More attacks (exfiltration)
- KDD Cup Challenge 1999
  - Based on DARPA 1998
    - Similar to netflow data

Public Data Sets

- DARPA 2000
  - DoS https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-datasets

Methods (I)

- Neural Networks
  - Cannady:
    - 10000 events, 3000 simulated by Internet Scanner and Satan
    - 93% accuracy
  - Lippmann and Cunningham
    - Analyzed telnet sessions
    - 80% detection with 1 false alarm per day

Methods (II)

- Neural Networks (cont'd)
  - – Bivens et al
    - DARPA 1999 added time windows
    - Predicts 100% of normal behavior
    - Some attacks not predicted
    - FAR (False Alarm Rate) 76%

Methods (III)

- (Fuzzy) Association Rules
  - IF (Bread AND Butter) → Milk (not fuzzy)
  - IF (X is A) → (Y is B) (A,B fuzzy sets)

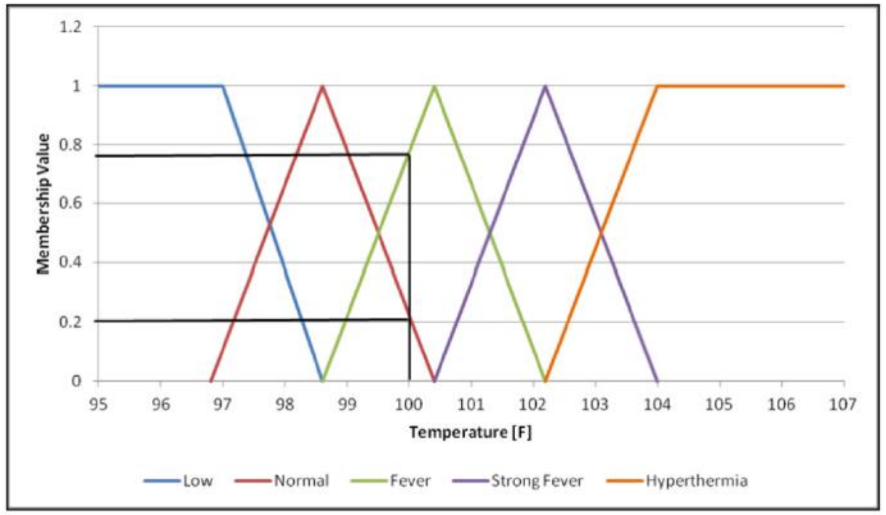


Fig. 2. Membership Functions for the Fuzzy Variable Human Body Temperature: Low, Normal, Fever, Strong Fever, and Hypothermia.

Methods (IV)

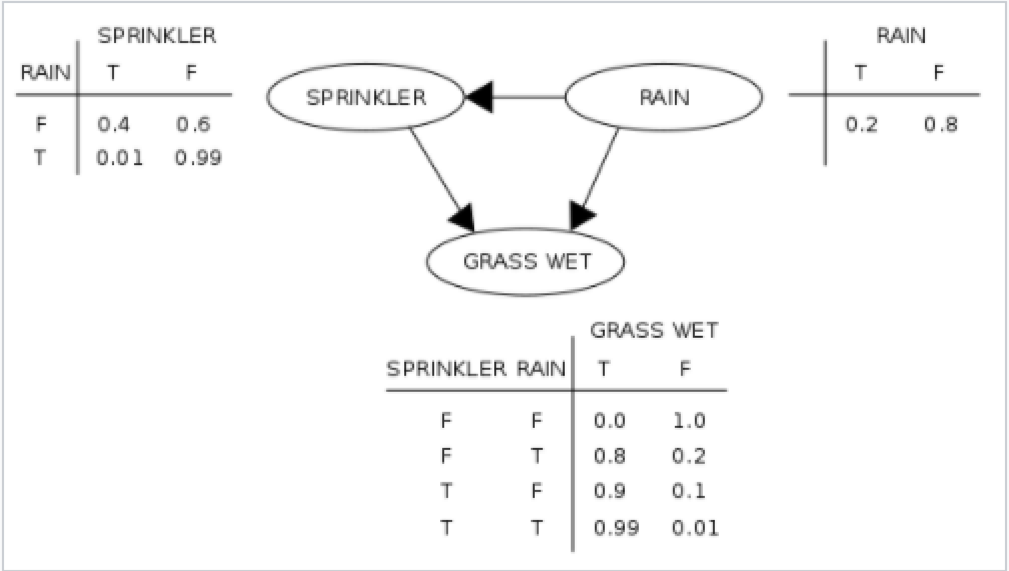
- Association Rule Mining
  - – Brahmi
    - DARPA 1998
    - E.g. (IF (service AND src_port AND dst_port AND num_conn) THEN attack_type) (4 dimensional)
    - 6 dimensional rules rates 99, 95, 75, 87% for DoS, Probe, U2R, R2L
    - Explainable- creates attack signatures

Methods (V)

- Association Rule Mining (cont'd)
  - Tajbakhsh et al
    - KDD 1999 data
    - Clustering approach
    - Association hyper-edge
      - {a,b} if average confidence of (a $\rightarrow$ b and b $\rightarrow$ a) is greater than 98% (with 50% confidence)
    - 100% accurate with 13% False Positive
      - Authors seem to get confused between recall and accuracy at this point

Methods (VI)

- Bayesian Network
  - Nodes are values of random variables
  - Edges are relationships between



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

A simple Bayesian network with conditional probability tables

Bayesian Network for IDS

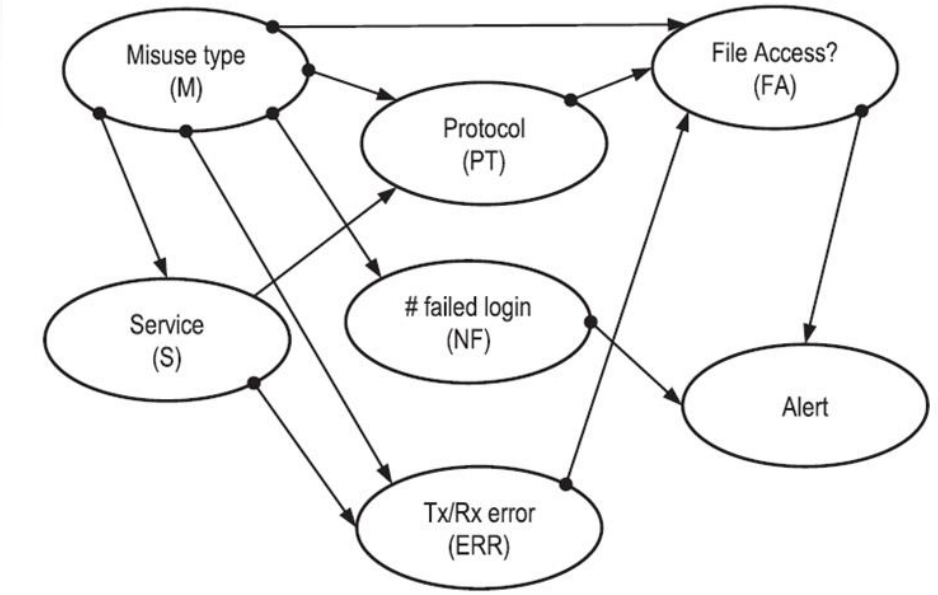
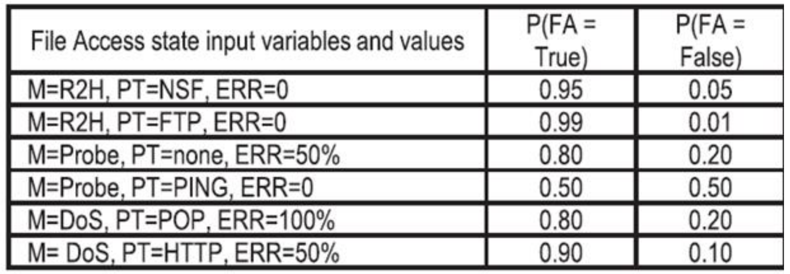


Fig. 3. Example Bayesian Network for Signature Detection.



File Access state input variables and values	P(FA = True)	P(FA = False)
M=R2H, PT=NSF, ERR=0	0.95	0.05
M=R2H, PT=FTP, ERR=0	0.99	0.01
M=Probe, PT=none, ERR=50%	0.80	0.20
M=Probe, PT=PING, ERR=0	0.50	0.50
M=DoS, PT=POP, ERR=100%	0.80	0.20
M= DoS, PT=HTTP, ERR=50%	0.90	0.10

Bayesian Net Methods (I)

- Livadas et al
  - IRC data 4 months at Dartmouth
  - Simulated data
  - 93% precision, false positive 1.39%
  - Naïve Bayes and C4.5 had higher precision (97%) and higher FP (1.47 and 8.05%)

Bayesian Net Methods (II)

- Kruegel et al
  - OS Kernel calls
  - 75% accuracy 0.2% FAR, 100% accuracy 0.1% FAR
    - Again authors seem to confuse accuracy with precision
- Benferhat et al.
  - DoS, no numeric results

Clustering Methods

- Simple Logfile Clustering Tool (SLCT) by Hendry/Yang
  - – Parameter M – percent of fixed features the cluster contains
    - 0 means everyone in on cluster
    - 100% means everyone in own cluster
    - M=97% yielded detection of 98% of attacks with 15% FAR
    - Can detect new attacks

SLCT (cont’d)

- Once you have clusters, are they attack or normal?
  - Evaluated based on % of times “words” appeared in normal vs. attack

SLCT Results

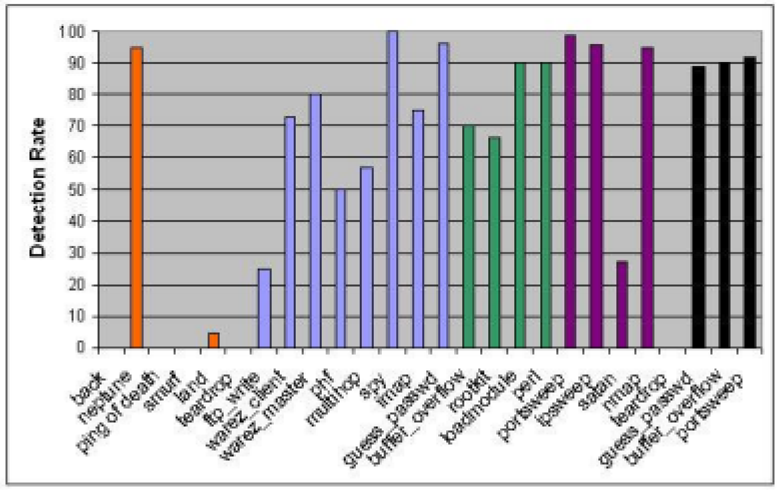


Figure 8. Detection Rate by Attack Type

Authors posit some poor results from too few instances of attack in data

Methods (VII)

- Decision Trees (e.g. C4.5, ID3)
  - Kruegel and Toth used these to show data from 1999 DARPA could be processed faster than Snort rules
    - Average 40.3% faster with 150 snort 2.0 rules– even more with full set of 1581 rules
  - EXPOSURE
    - DNS classification
      - Detected 98.5% of bad domains with FAR of 0.9%
      - Also detected 3117 new malicious domains!

Methods (VIII)

- Ensemble Learning
  - Use multiple weak learners to build a strong one
    - E.g. Random forest – create decision trees from randomly picked attributes

Ensemble Learning Results (I)

- Zhang et al
  - Outlier detection + classifier (Random Forest)
  - KDD 1999
  - 1.92% error with 0.01% FAR on original
- Gharibian and Ghorbani
  - Random Forest
    - Accuracies 97%, 76%, 5%, 35% for DoS, Scan, R2L and U2R

Ensemble Learning Results (II)

- Mukkamala et al
  - – 3 neural nets with majority voting
  - – DARPA 1998 data 80% attacks
  - – 99.71%, 99.85%, 99.97%, 76% and 100% for Normal, Scan, DoS, U2R, R2L
- Bilge et al DISCLOSURE
  - – Random Forests on netflow
    - True positive 65%, False positive 1%

Methods (IX)

- Genetic Algorithms/Programming
  - Evolutionary process
  - GA on bit strings, crossover and mutation simple
  - GP trees/programming blocks- more complex

Genetic Algorithm Results (I)

- Li
  - DARPA IDS
  - Chromosome contained source/dest IP addr & port, duration, protocol, # of bytes each direction, state of connection
  - Fitness function weights some parts more than others
  - No accuracy provided

Genetic Algorithm Results (II)

- Khan
  - KDD 1999 data subset
  - Used principal component analysis to identify 8 attributes to focus on
  - Population of 10 individuals, 1 chosen to classify
  - Accuracy 93.45%, FAR 10.8% normal
  - Accuracy 94.19%, FAR 2.75% attack
    - Accuracy => Precision?

Genetic Programming Results (III)

- Abraham et al
- Linear Genetic Programming, Multi-Expression Programming and Gene Expression Programming
- DARPA 1998 IDS
- FAR from 0 to 5.7%

Genetic Programming Results (IV)

- Hansen et al
- Homologous crossover to prevent program growth
- KDD 1999 subset
  - Better on subset, but maybe not on whole?

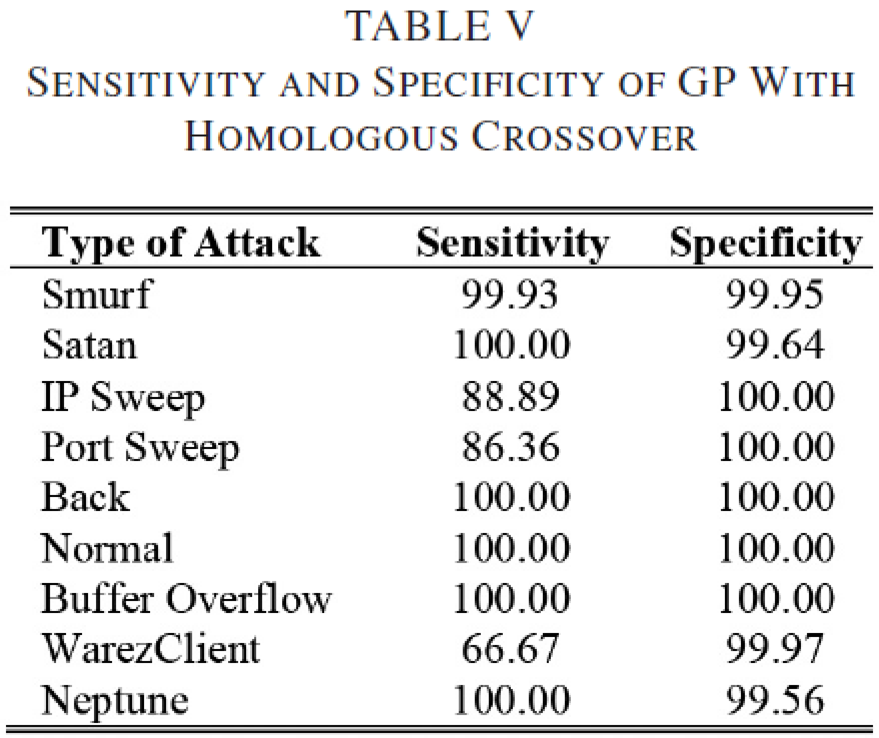


TABLE V
SENSITIVITY AND SPECIFICITY OF GP WITH HOMOLOGOUS CROSSOVER

Type of Attack	Sensitivity	Specificity
Smurf	99.93	99.95
Satan	100.00	99.64
IP Sweep	88.89	100.00
Port Sweep	86.36	100.00
Back	100.00	100.00
Normal	100.00	100.00
Buffer Overflow	100.00	100.00
WarezClient	66.67	99.97
Neptune	100.00	99.56

Genetic Programming Results (V)

- Subset DARPA IDS (1 day)
  - Training: 1000 connection records, 8 attack types
  - Test: 1 day, 10 attack types
- 10000 runs of GP – average reported
  - FAR 0.41%, Detected 57%
- Best detects 100% with FAR 1.4-1.8%

Methods (X)

- Hidden Markov Models
  - Set of states with transition probabilities

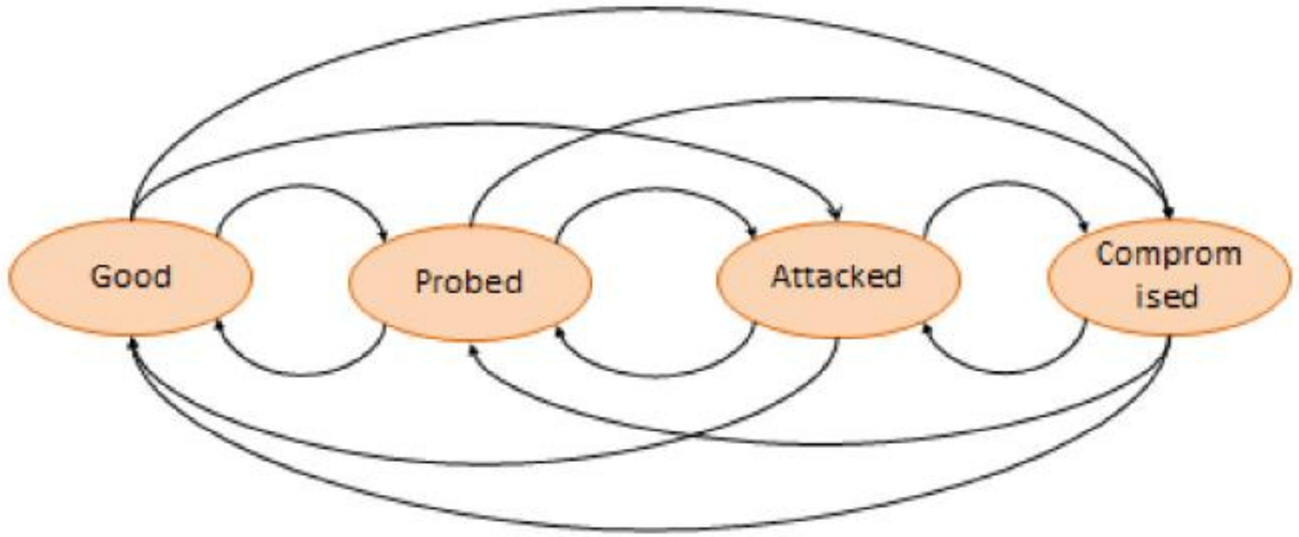


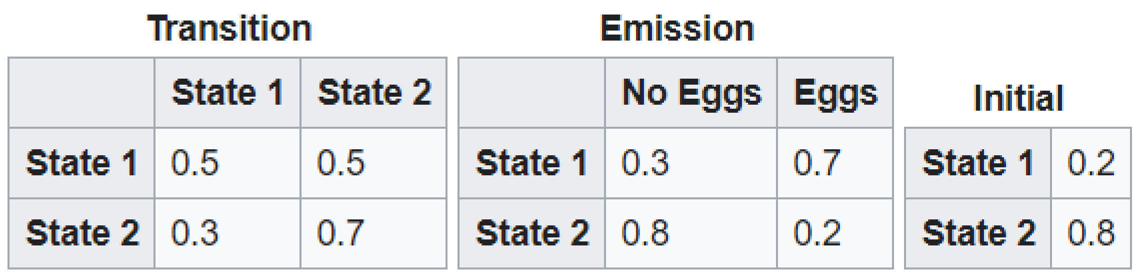
Fig. 5. An Example Hidden Markov Model.

Hidden Markov Models

- Often number of states is guessed and states have no discernable meaning
- Wikipedia (Baum Welch) chicken egg example
  - Guess two states
  - Don’t know initial state, probabilities of switching states, or probability of laying eggs in either state

HMM cont'd

- Guess everything!



Transition

	State 1	State 2
State 1	0.5	0.5
State 2	0.3	0.7

Emission

	No Eggs	Eggs
State 1	0.3	0.7
State 2	0.8	0.2

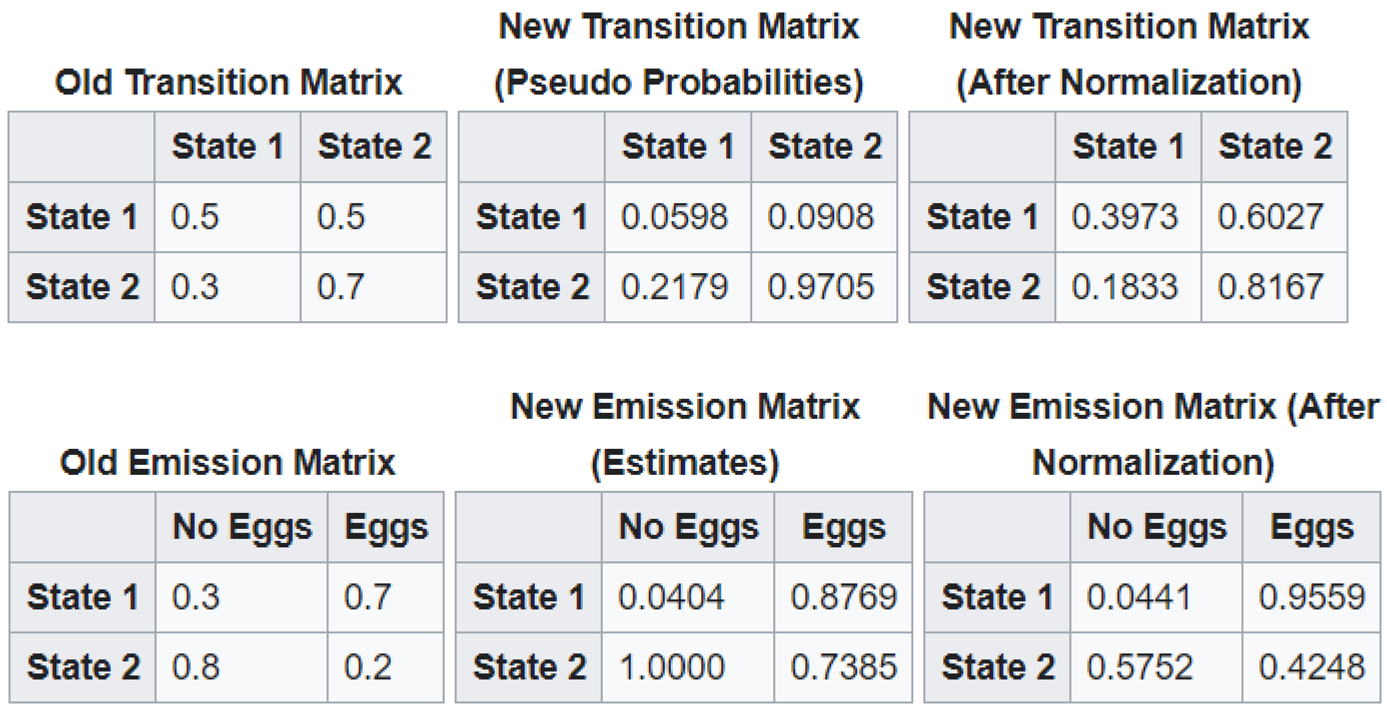
Initial

State 1	0.2
State 2	0.8

- Now look at observations
  - N, N, N, N, N, E, E, N, N, N
  - Resulting transitions
    - NN, NN, NN, NN, NE, EE, EN, NN, NN

HMM cont’d

- Do math to update matrices based on observations, repeat until converges



Old Transition Matrix

	State 1	State 2
State 1	0.5	0.5
State 2	0.3	0.7

New Transition Matrix (Pseudo Probabilities)

	State 1	State 2
State 1	0.0598	0.0908
State 2	0.2179	0.9705

New Transition Matrix (After Normalization)

	State 1	State 2
State 1	0.3973	0.6027
State 2	0.1833	0.8167

Old Emission Matrix

	No Eggs	Eggs
State 1	0.3	0.7
State 2	0.8	0.2

New Emission Matrix (Estimates)

	No Eggs	Eggs
State 1	0.0404	0.8769
State 2	1.0000	0.7385

New Emission Matrix (After Normalization)

	No Eggs	Eggs
State 1	0.0441	0.9559
State 2	0.5752	0.4248

HMM Results (I)

- Ariu et al
  - – DARPA 1999 dataset
  - – Multiple HMM classifiers
  - – Area under ROC curve 0.915 to 0.976
  - – FP rate 0.1%, detects 85%
  - – FP rate 0.01%, detects > 70%

HMM Results (II)

- Joshi and Phoha
  - 5 states, 6 observation symbols
  - All states interconnected
  - Used Baum-Welch method to estimate parameters
  - KDD 1999, 5 of 41 features
  - Recall 79%

Methods

- Inductive Learning
  - Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
  - Separate and Conquer approach
    - Create rule that covers a maximal set of examples (e.g. # password guesses > 5)
    - Remove all those correctly labeled from the dataset
    - Repeat until training set empty, or stopping criteria met
  - Competes with C4.5 Decision Trees

Incremental Reduced Error Pruning

```
Initialize E to the instance set
Until E is empty do
    Split E into Grow and Prune in the ratio 2:1
    For each class C for which Grow contains an instance
        Use basic covering algorithm to create best perfect rule for C
        Calculate w(R): worth of rule on Prune
            and w(R-): worth of rule with final condition omitted
        If w(R-) < w(R), prune rule and repeat previous step
    From the rules for the different classes, select the one that’s
        worth most (i.e. with largest w(R))
    Print the rule
    Remove the instances covered by rule from E
Continue
```

RIPPER mods

Order classes according to increasing prevalence

    (C1,....,Ck)
    find rule set to separate C1 from other classes
    IREP(Pos=C1,Neg=C2,...,Ck)
    remove all instances learned by rule set
    find rule set to separate C2 from C3,...,Ck
    ...
    Ck remains as default class

RIPPER results

- Lee et al
  - DARPA 1998 telnet connections
  - 38 attacks, 14 in test only
  - New attack detected 5.9% to 96.7% of the time
  - Old attacks detected 60% to 97%

RIPPER results

- Fan et al
  - DARPA 1998 data set
  - Artificial anomaly generator
  - 94% detection with 2% FAR (too big for packets)

Methods (XI)

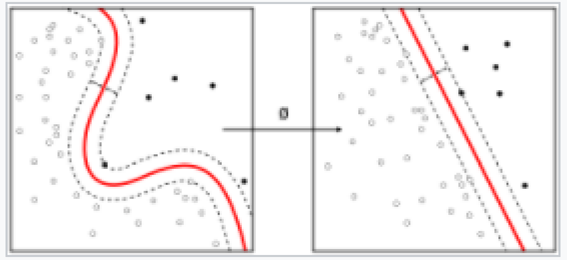
- Naïve Bayes
  - Conditional probabilities
  - Optimal if features conditionally independent (not likely)
  - Panda and Patra
    - KDD 1999, FAR 3% detection rate 90-99%
  - Amor et al
    - KDD 1999, detection rates 97%, 96%, 9%, 12%, 88%. FAR ? (< 3 %)

Methods (XII)

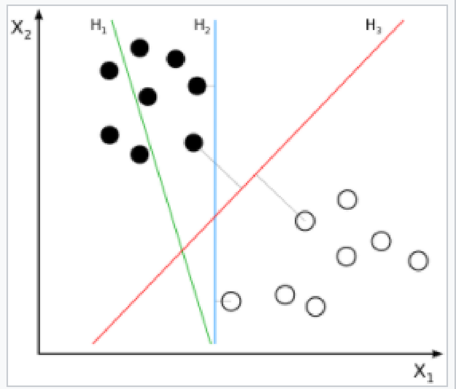
- Sequential Pattern Mining
  - Find statistically relevant patterns in sequenced data (e.g. with timestamps)
  - Li et al
    - Discover multi-stage attack patterns
    - DARPA 1999 and DARPA 2000 datasets
    - Detected 93% of attacks in 20 seconds
    - 84% in another scenario

Methods (XIII)

- Support Vector Machine (SVM)
  - Split points using hyperplanes with max distance between hyperplane and data points
  - Best when # of features is high
  - When non-separable, add "cost" for overlapping data points
  - Can use kernel to linearize data



Kernel machine



H_1 does not separate the classes. H_2 does, but only with a small margin. H_3 separates them with the maximal margin.

SVM Results (I)

- Li et al
  - RBF kernel (radial basis function)
  - KDD 1999 data set
  - 19 of 41 features used
  - 98% accuracy overall (53% for U2R)
  - Subset of training set (ant colony optimization)

SVM Results (II)

- Amiri et al
  - Random sample of 7000 instances (from 5M) for each of 5 classes
  - Mutual information to reduce features
    - If there is lots of mutual info between X and Y, maybe we don't need Y
  - 5 classifiers per category
  - 99% on 4 of 5 classes, 93% on U2R

SVM Results (III)

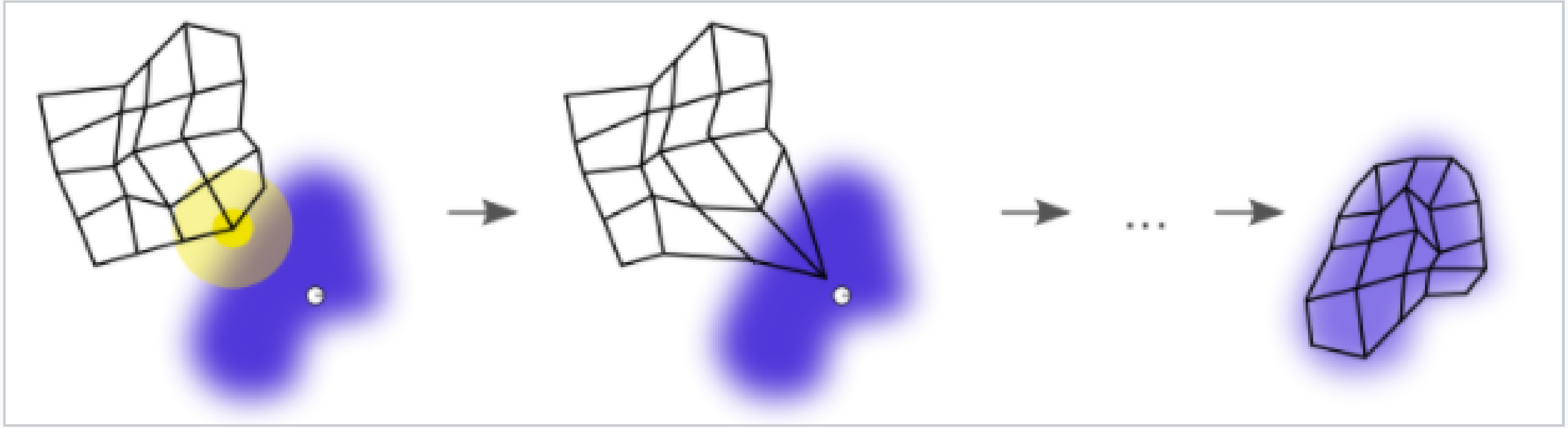
- Hu et al
  - Basic Security Module portions of DARPA 1998
  - 75% detection no false alarms, 100% with 3% FAR

Enhanced SVM

- Shon and Moon

1. create profile of normal with Self-Organized Feature Map (SOFM)
2. Packet filter with passive TCP filtering (PTF) to reject incomplete or non-compliant traffic (looking for normal)
3. Use genetic alg to extract optimized info from raw packets
4. Use temporal flow for data preprocessing

SOFM



An illustration of the training of a self-organizing map. The blue blob is the distribution of the training data, and the small white disc is the current training datum drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node (highlighted in yellow) which is nearest to the training datum is selected. It is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).

Enhanced SVM Results

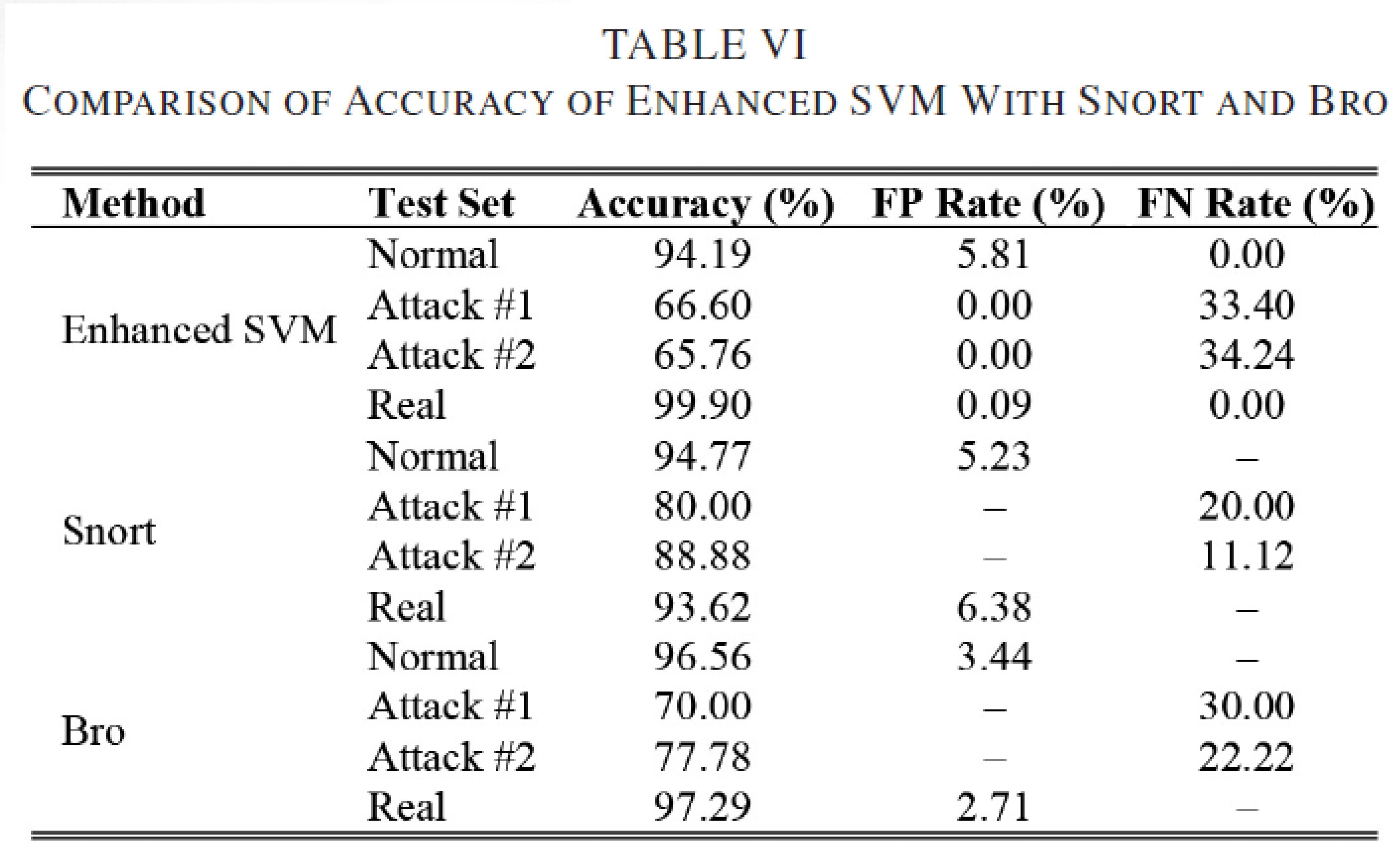


TABLE VI
COMPARISON OF ACCURACY OF ENHANCED SVM WITH SNORT AND BRO

Method	Test Set	Accuracy (%)	FP Rate (%)	FN Rate (%)
Enhanced SVM	Normal	94.19	5.81	0.00
	Attack #1	66.60	0.00	33.40
	Attack #2	65.76	0.00	34.24
	Real	99.90	0.09	0.00
Snort	Normal	94.77	5.23	–
	Attack #1	80.00	–	20.00
	Attack #2	88.88	–	11.12
	Real	93.62	6.38	–
Bro	Normal	96.56	3.44	–
	Attack #1	70.00	–	30.00
	Attack #2	77.78	–	22.22
	Real	97.29	2.71	–

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Training-Resistant Anomaly Detection System

Dr. Martin “Doc” Carlisle

Premise

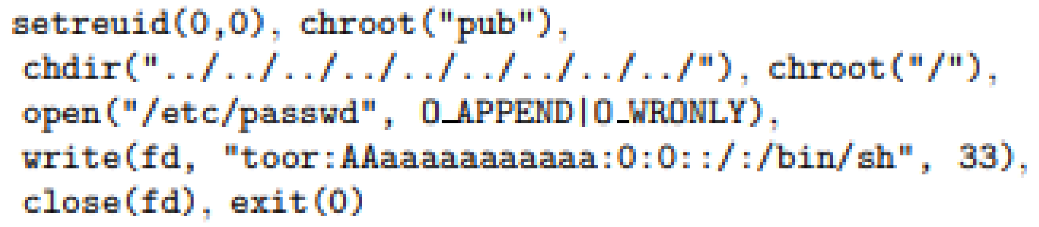
- Probably need unsupervised learning to solve IDS
- Training attack
  - Malicious actors may inject traffic to “teach” the IDS to accept dangerous as normal
- Detect by considering input at multiple time resolutions

Related Work

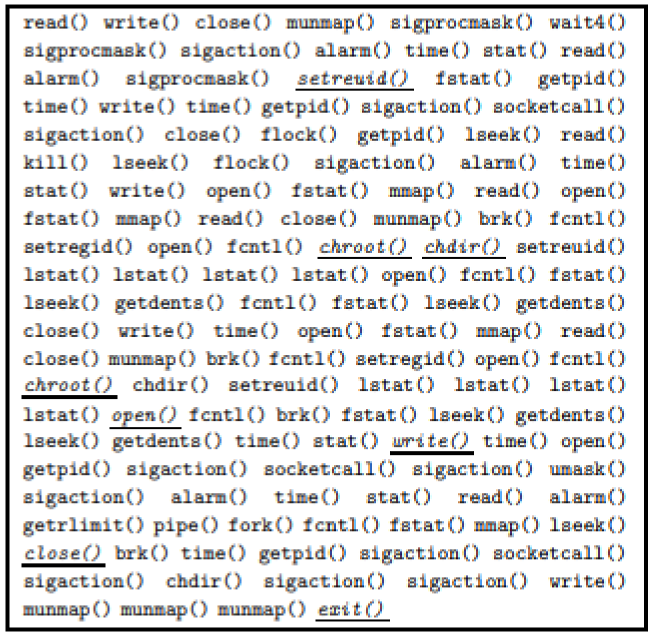
- Clustering algorithms do unsupervised learning to detect anomalous behavior
- Mimicry attack
  - Attempt to evade IDS by stealth
- Yen and Reiter – monitor similar behavior of hosts across network
  - Only for malware spread, not targeted attack

Mimicry attack in Library Calls

- Original attack vs. mimic one



```
setreuid(0,0), chroot("pub"),
 chdir("../../../../../../../../../"), chroot("/"),
 open("/etc/passwd", O_APPEND|O_WRONLY),
 write(fd, "toor:AAaaaaaaaaaaa:0:0::/:/bin/sh", 33),
 close(fd), exit(0)
```



```
read() write() close() munmap() sigprocmask() wait4()
sigprocmask() sigaction() alarm() time() stat() read()
alarm() sigprocmask() setreuid() fstat() getpid()
time() write() time() getpid() sigaction() socketcall()
sigaction() close() flock() getpid() lseek() read()
kill() lseek() flock() sigaction() alarm() time()
stat() write() open() fstat() mmap() read() open()
fstat() mmap() read() close() munmap() brk() fcntl()
setregid() open() fcntl() chroot() chdir() setreuid()
lstat() lstat() lstat() lstat() open() fcntl() fstat()
lseek() getdents() fcntl() fstat() lseek() getdents()
close() write() time() open() fstat() mmap() read()
close() munmap() brk() fcntl() setregid() open() fcntl()
chroot() chdir() setreuid() lstat() lstat() lstat()
lstat() open() fcntl() brk() fstat() lseek() getdents()
lseek() getdents() time() stat() write() time() open()
getpid() sigaction() socketcall() sigaction() umask()
sigaction() alarm() time() stat() read() alarm()
getrlimit() pipe() fork() fcntl() fstat() mmap() lseek()
close() brk() time() getpid() sigaction() socketcall()
sigaction() chdir() sigaction() sigaction() write()
munmap() munmap() munmap() exit()
```

Attack Scenarios

- DoS
- Brute Forcing
- Network Scans
- Routing attacks
- Worms

Detecting these

- Measure between pairs of hosts
  - Number of bytes/time period
  - Average packet size
  - # of concurrent connections
  - Pause since last packet

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Attack Scenarios

- DoS

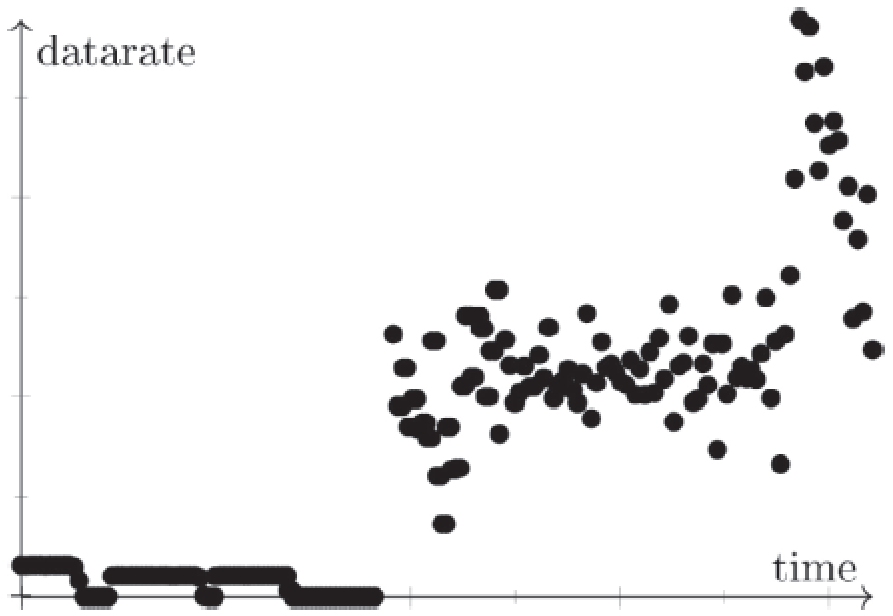


Fig. 1 – Illustration of a typical denial-of-service attack. Note the abrupt "jumps" for the measured data rate. This example is based on the data sets recorded by Garcia et al. (2014).

Training Attacks

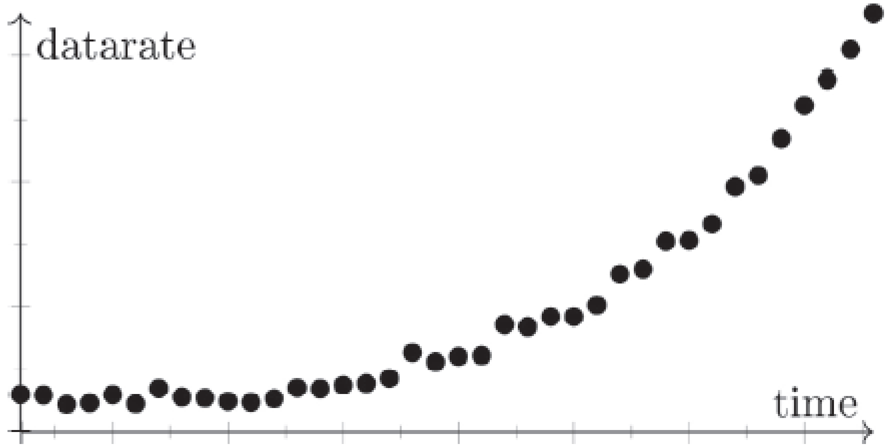


Fig. 2 – Illustration of the training attack over time, for the case of the data rate. An attacker proceeds by progressively injecting more and more packets until he eventually reaches the desired critical threshold.

Stealthy Training Attacks

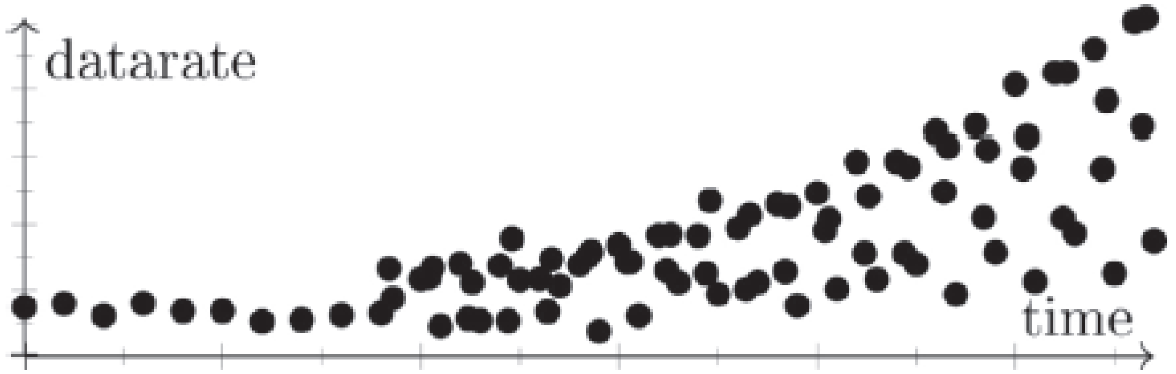


Fig. 3 – Illustration of the *stealthy* training attack over time. Instead of only increasing the traffic load, an attacker creates enough "normal" data in-between, which outweigh (and thus hide) the malicious traffic.

IDS Techniques (I)

- Threshold and metric
  - Fixed: requires human effort, unable to adapt
  - Average and variance: have difficulty with inhomogeneous data
    - Subject to training attack

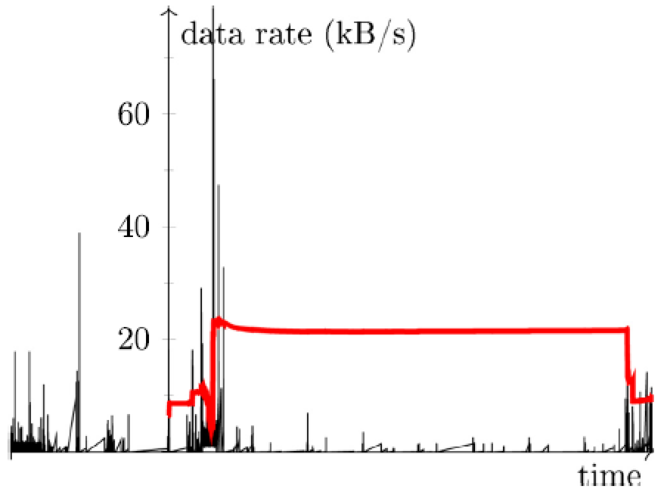


Fig. 4 – Illustration of typical data rates for HTTP traffic (thin black line). Note how the $\mu + 3 \cdot \sigma$ threshold value (thick red line) is not a good descriptor of "normal" traffic and thus a bad candidate for detecting outliers in the traffic. This is due to the fact that HTTP traffic is not even closely normally distributed. (For interpretation of the references to colour in Figure 4, the reader is referred to the web version of this article.)

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

IDS Techniques (II)

- Stream clustering

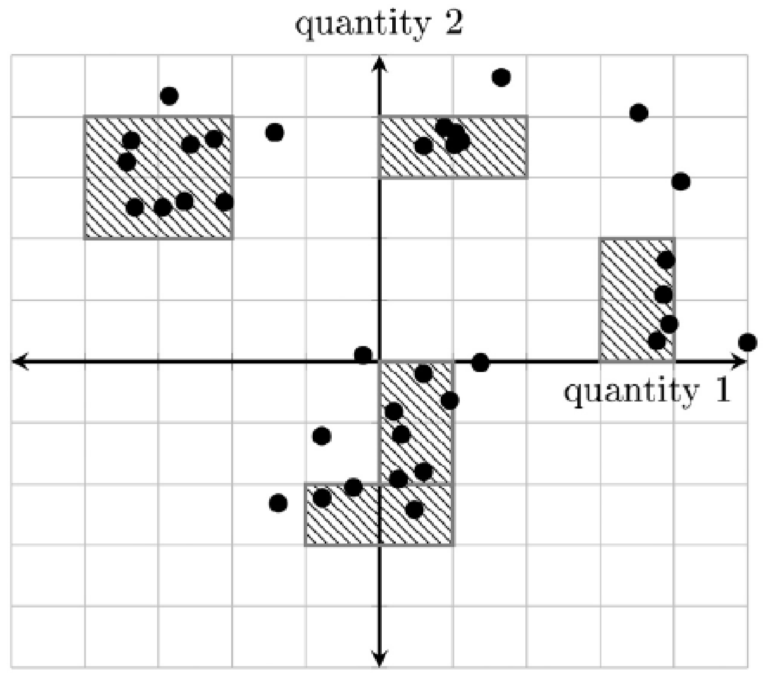


Fig. 5 – Example of a two-dimensional state space divided into a grid, at a particular time t. Bullets denote data points; dense cells are hatched. In this example, there exist 4 clusters at time t.

Proposed IDS

- Alert raised when new cluster created

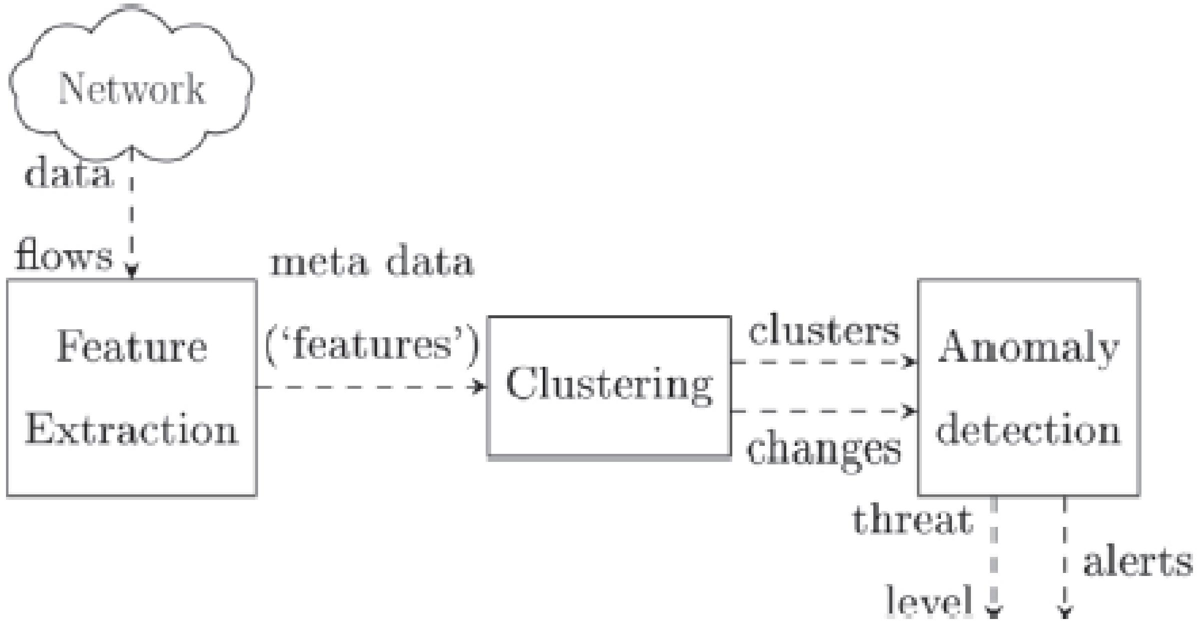


Fig. 6 – Workflow of the intrusion detection system.

Detecting Training Attack (I)

- Deploy clones of IDS with different parameters
- Slow evolution appears as deletion/creation of cluster

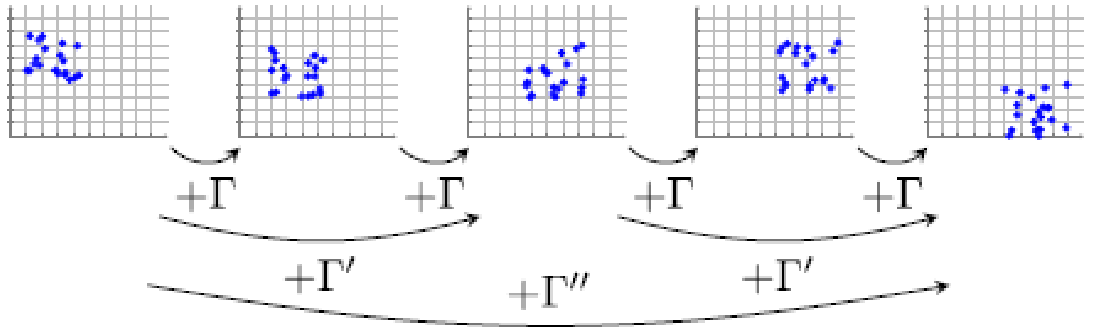


Fig. 7 – The several steps of the training attack and how they appear at different time scales.

Detecting Training Attack (II)

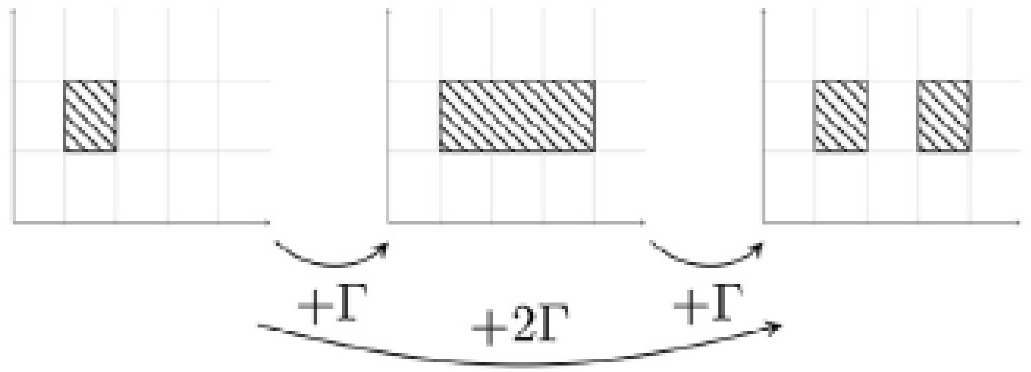


Fig. 8 – The two (legitimate) steps that the training attack consists of. Dense cells are hatched. If the intermediate clustering is omitted, the situation will look as if a new cluster is created, and an alert will be raised.

Detecting Stealthy Training Attack

- Classic training attack moves clusters
- Stealthy one enlarges them
- To detect, add another instance of IDS on size of clusters of actual IDS

Evaluation

- Interestingly, authors used Forensic exercises as datasets
  - Digital Corpora M57 patents
    - Exfiltration of data from corporate network
  - Digital Corpora Nitroba University
    - Email harassment scenario
- And 4SICS geek lounge SCADA network capture
  - But they added artificial attack traffic to it

Network attack detection

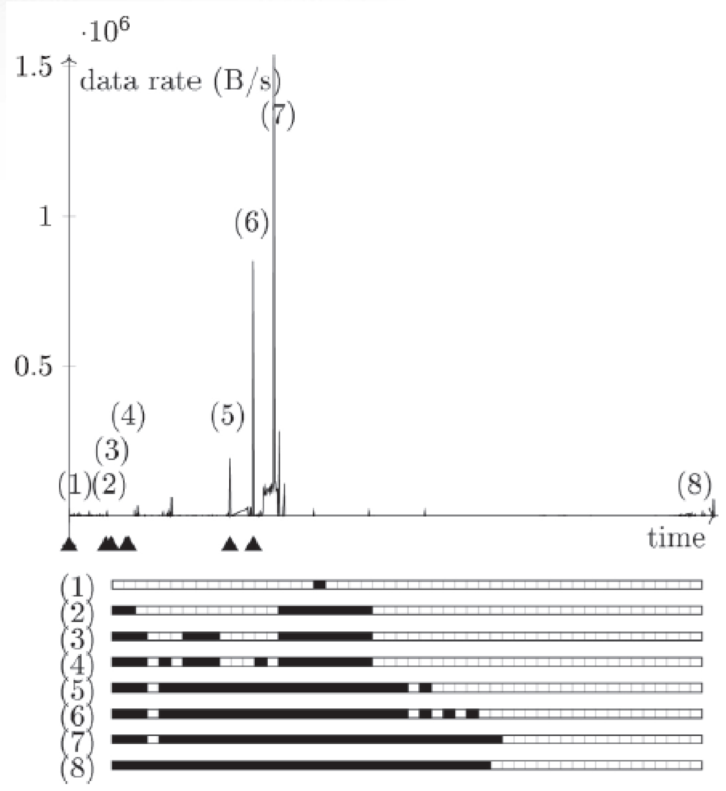


Fig. 9 – Successful detection of unusually high out-going traffic for one of the hosts in the Digital Corpora, 2009 data set. The triangles denote the times when new clusters were created (thus alerts raised). Below the figure, the clusters are explicitly drawn for the snapshots (1) to (8): boxes represent the cells in the one-dimensional state space; black boxes denote dense cells.

Training attack detection

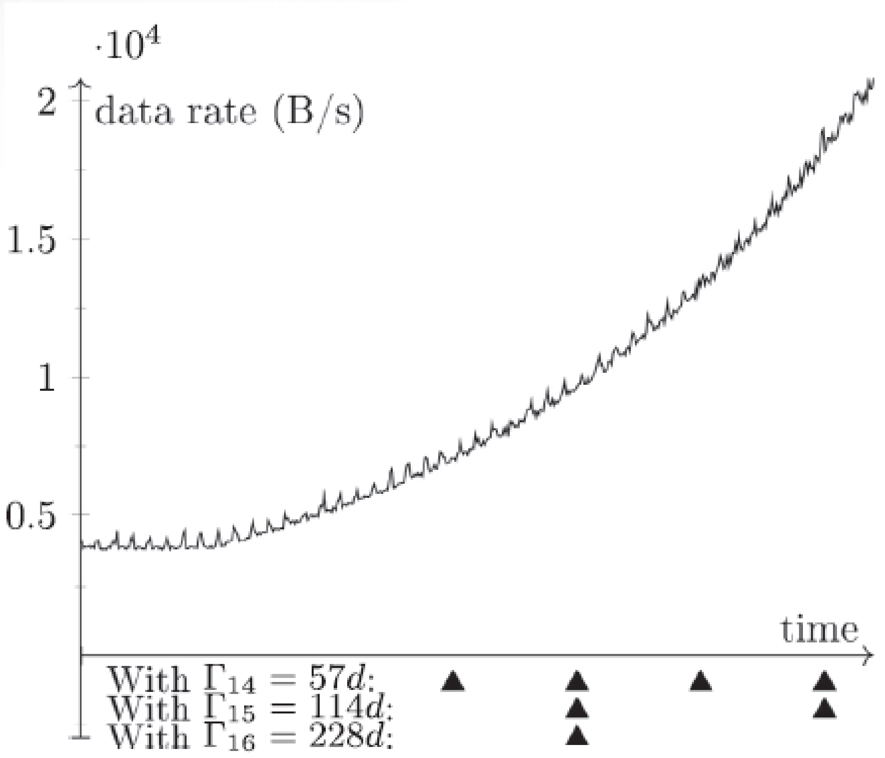


Fig. 12 – The training attack combined with the Netresec AB, 2015 data set. The triangles denote the times when new clusters were created (thus alerts raised) for the instances with Γ = 57 *d*, 114 *d*, 228 *d*, respectively.

Stealthy Attack Evaluation

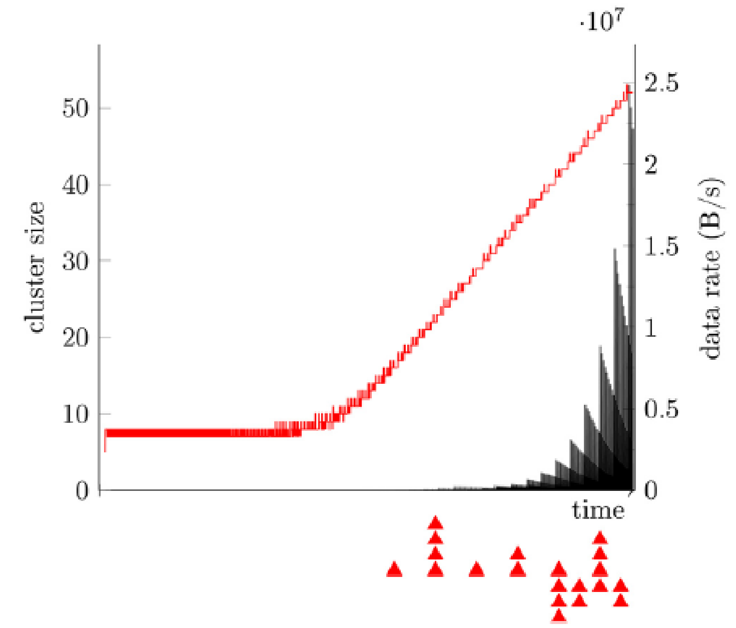


Fig. 13 – The stealthy training attack combined with the Netresec AB, 2015 data set. The right graph depicts the data rate as induced by the attack, while the left graph depicts the evolution of the cluster size. Although no alert is raised for the behaviour of the *data rate*, the dual IDS does identify the increase of the *cluster size* (alerts marked with red triangles). (For interpretation of the references to colour in Figure 13, the reader is referred to the web version of this article.)