Network Security & Machine Learning

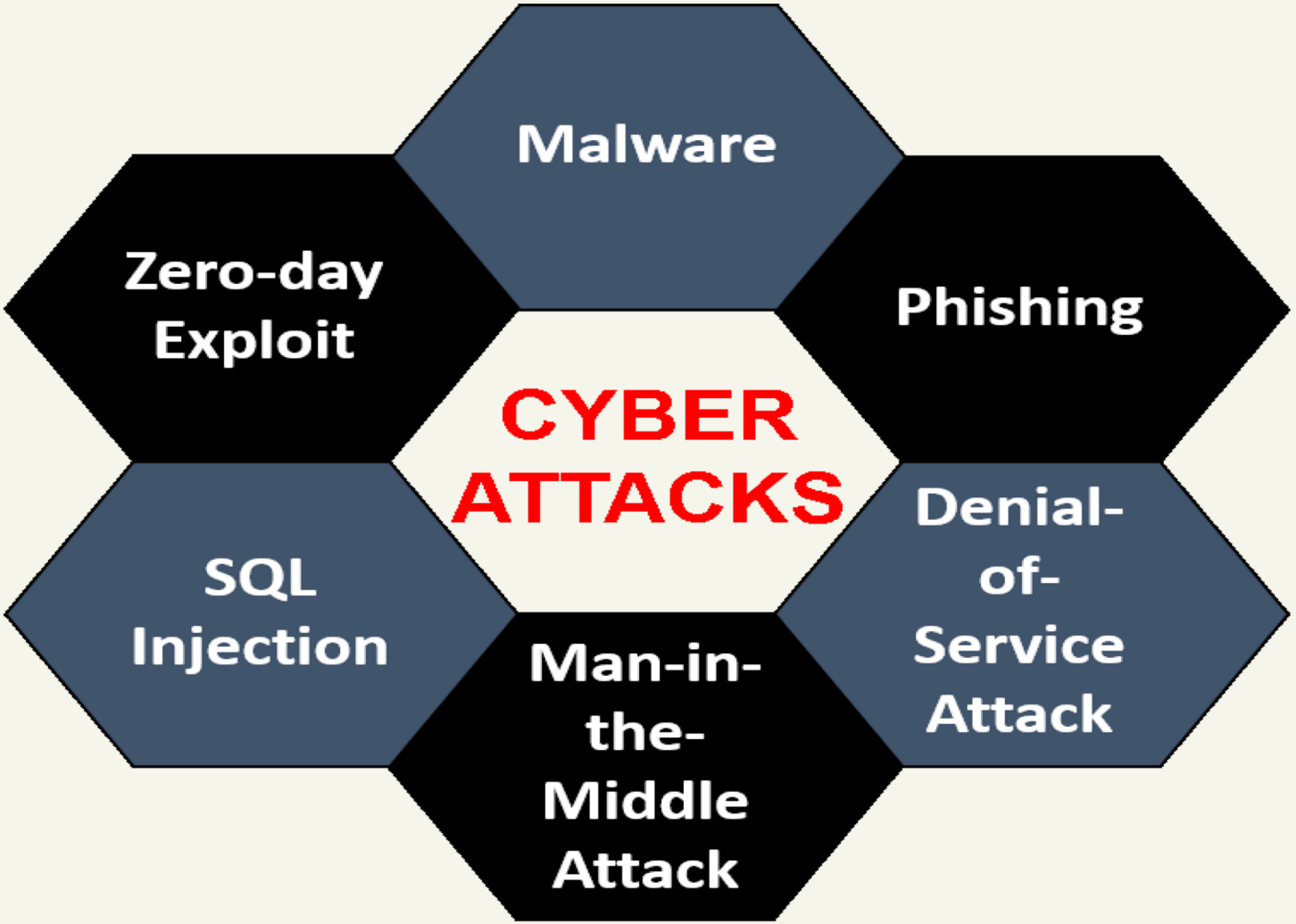
CMPE 209 Network Security

Younghee Park

Content

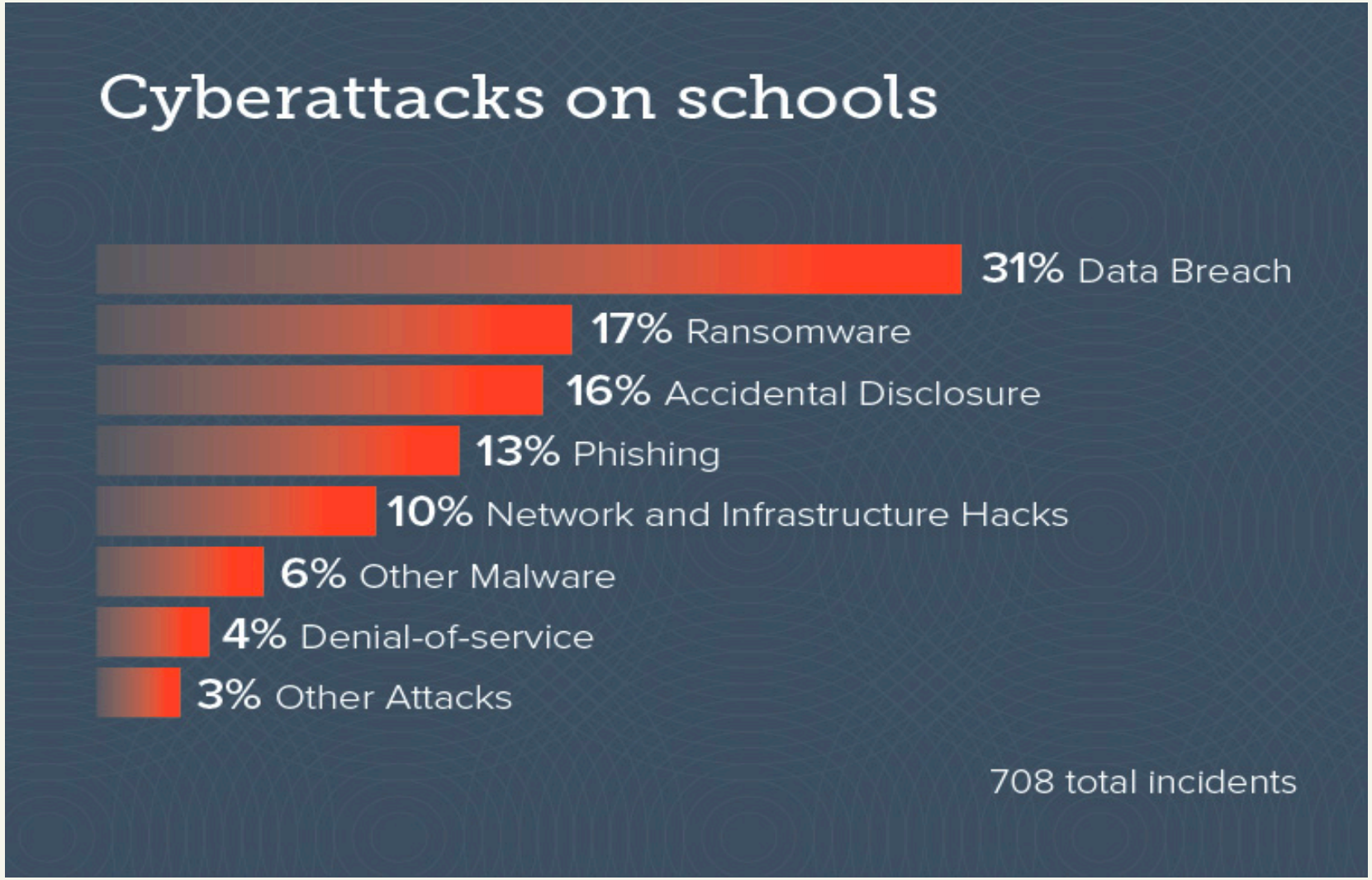
- Basic cybersecurity concept
- Network security
- Machine learning
- ML-based network intrusion detection systems

Cyberattacks



Source: Webservers on Feb. 20, 2019

Cyberattacks on schools



Source: Barracuda on Oct. 31, 2019

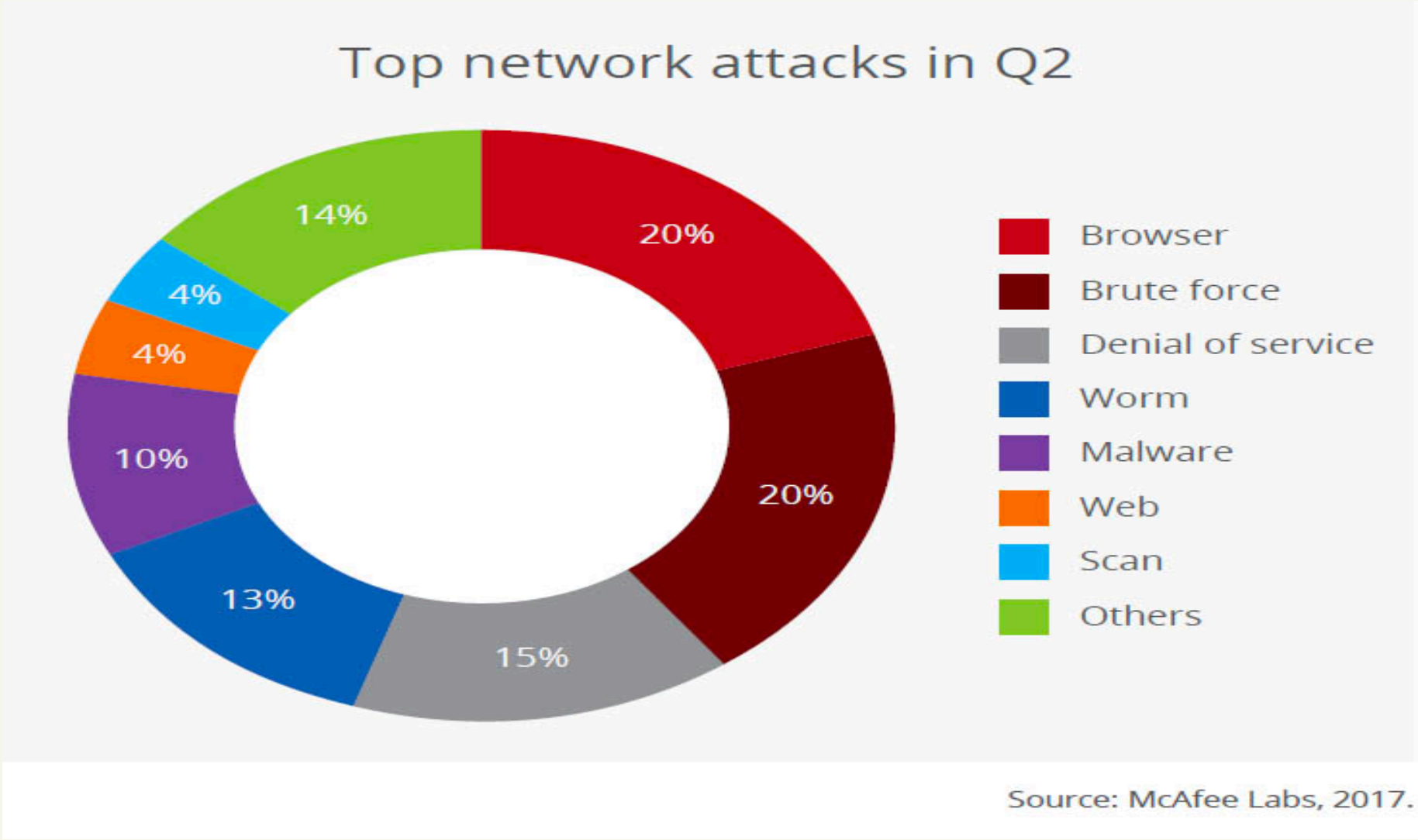
Security Services

Confidentiality	**Authentication**
Integrity	**Non-repudiation**
Access Control	**Monitor & Response**

Cyber Threats or Attacks

Source: European Union Agency for CyberSecurity

Network Attacks



Source: Macfee Labs, 2017

Network Intrusion Detection System

- Network Intrusion Detection System (NIDS)
  - **IDS**: SW or HW designed to monitor and analyze and respond to events occurring in a computer system or network for signs of possible incidents of violation in security policies.
  - It needs **traffic analysis**
  - More advanced packet filters than conventional firewalls
  - Analyze **payload of each packet** with predefined signature or anomaly and flags the traffic as good or malicious
  - **Malicious packets logged** for further analyses by network administrators or security experts.

IDS

Signature based IDS

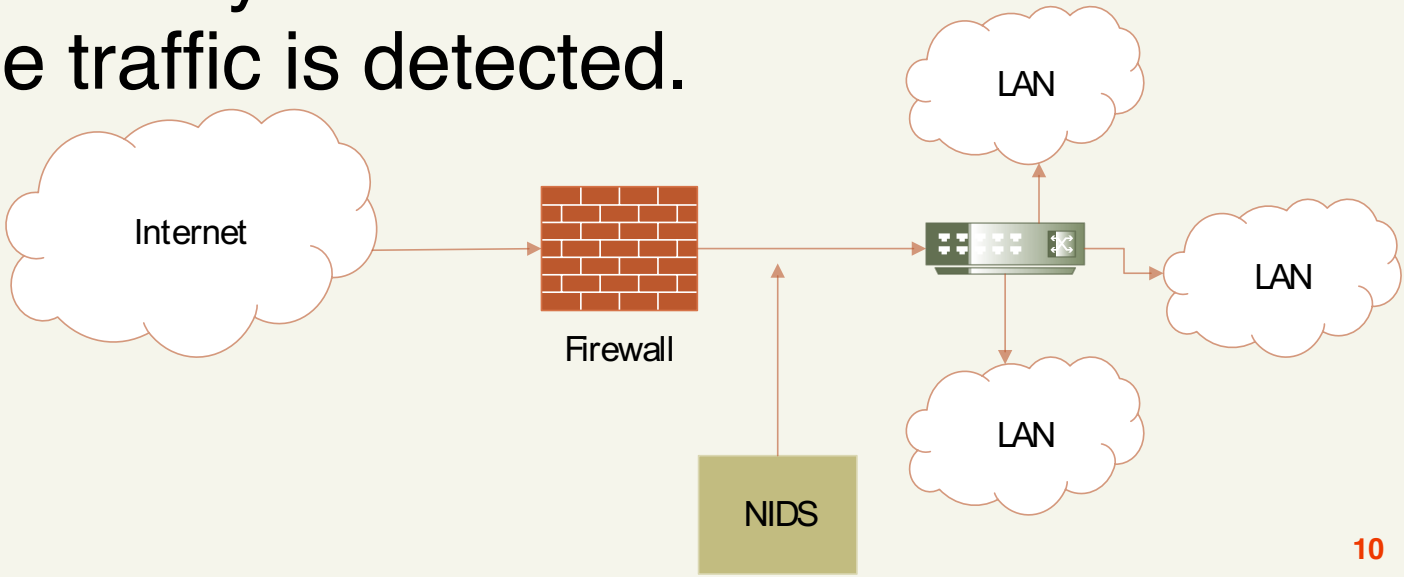
Anomaly based IDS

Host based IDS

Network based IDS

Network Intrusion Detection System

- Network Intrusion detection systems (NIDSs) are software or hardware systems that automate the process of monitoring the events occurring in a network, and analyzing them for signs of security problems.
- NIDSs provide a layer of defense which monitors network, and alerts system administrators when potential hostile traffic is detected.



Security Companies

RSA

SVCSI SILICON VALLEY CYBERSECURITY INSTITUTE

McAfee Together is power.

Check Point SOFTWARE TECHNOLOGIES LTD.

ixia A Keysight Business

THALES

tripwire

paloalto NETWORKS

SOPHOS Security made simple.

Symantec

FireEye

TREND MICRO

FORCEPOINT

bitglass

FORTINET

Google

JUNIPER NETWORKS

intel

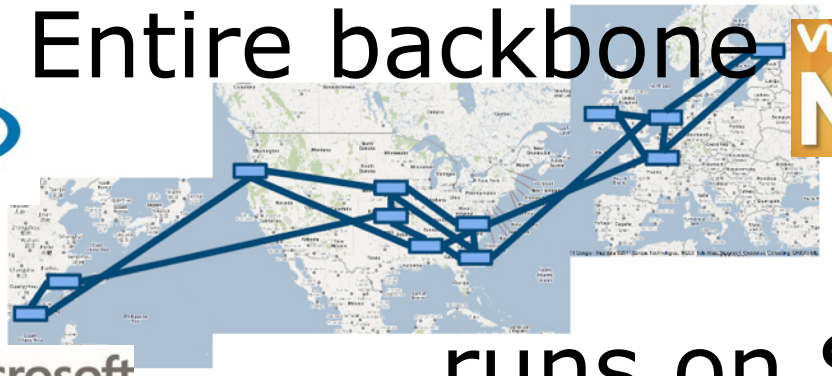
Akamai

CISCO

IBM

Microsoft

Entire backbone



vmware NSX

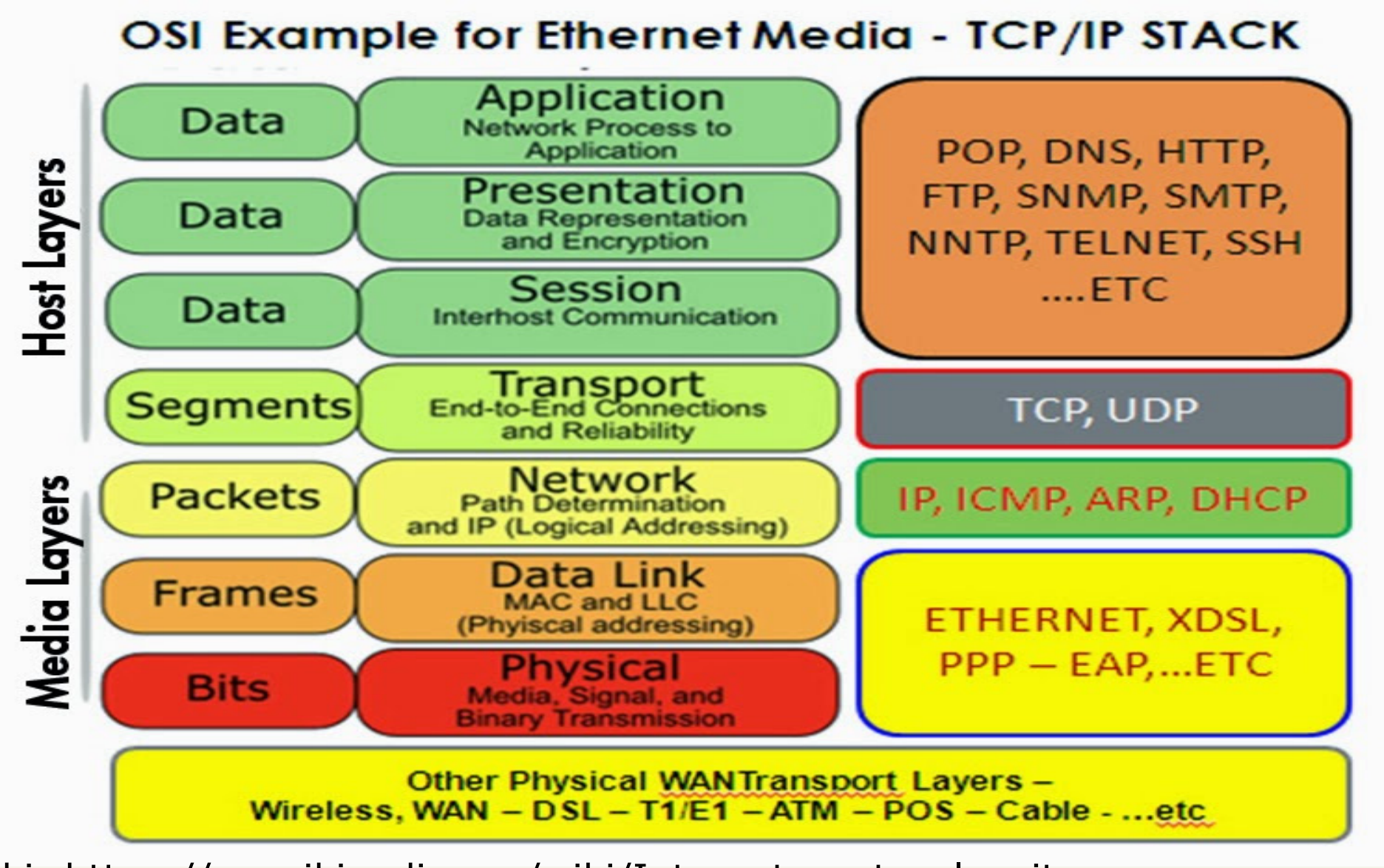
runs on SDN

Traffic Analysis

Traffic Analysis

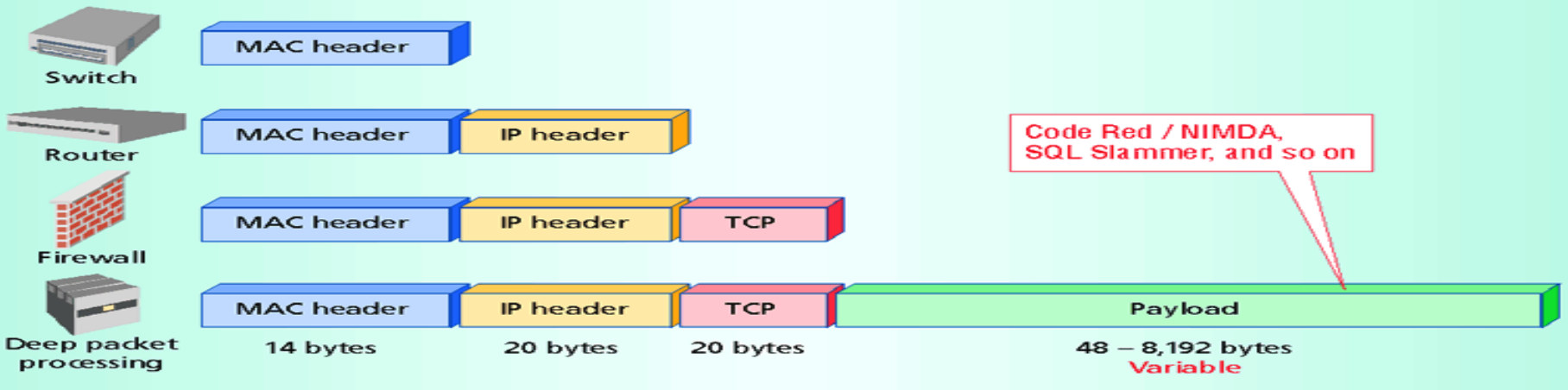
- Detection for abnormal behavior in networks
  - Abnormal packets, unexpected traffic
  - Load balancer issues
  - Network slow performance due to congestion/retransmission
- Network forensics to collect evidence, to handle incident, to trace attacks
  - Develop IPS/IDS signatures
- Traffic Capturing Tools: TCPdump, Tshark, Wireshark, etc.

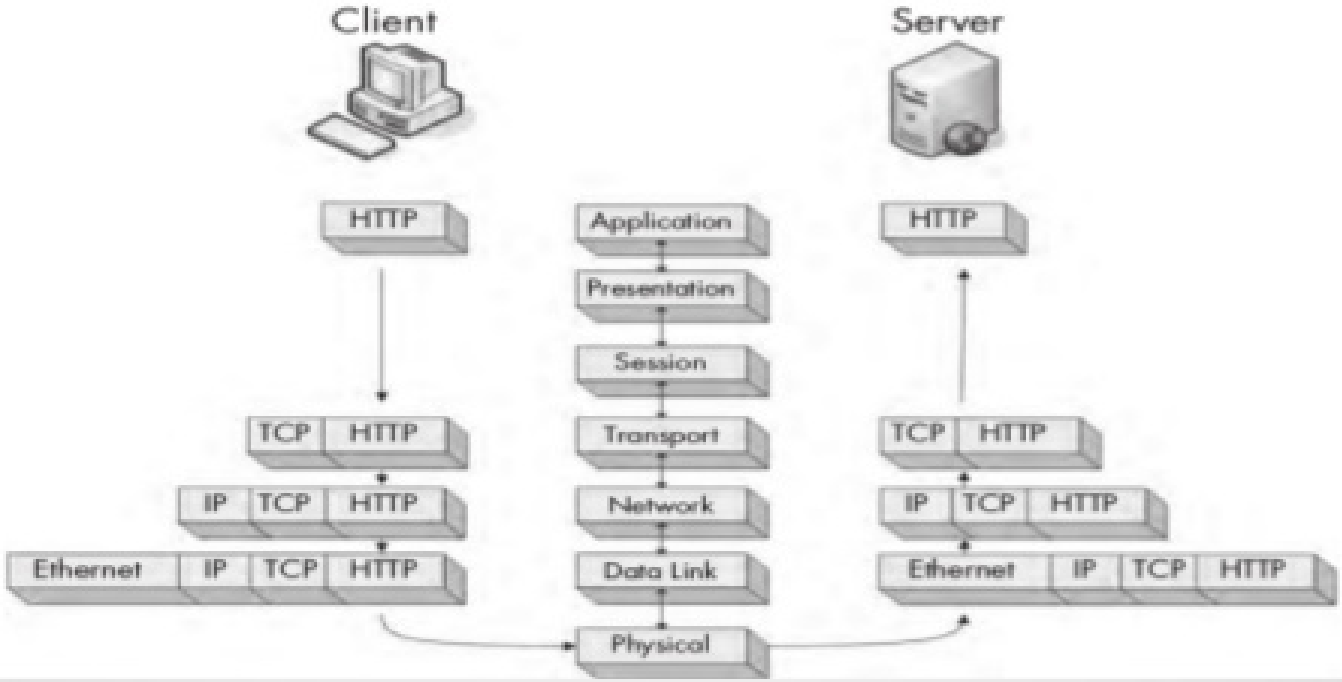
Internet Protocol



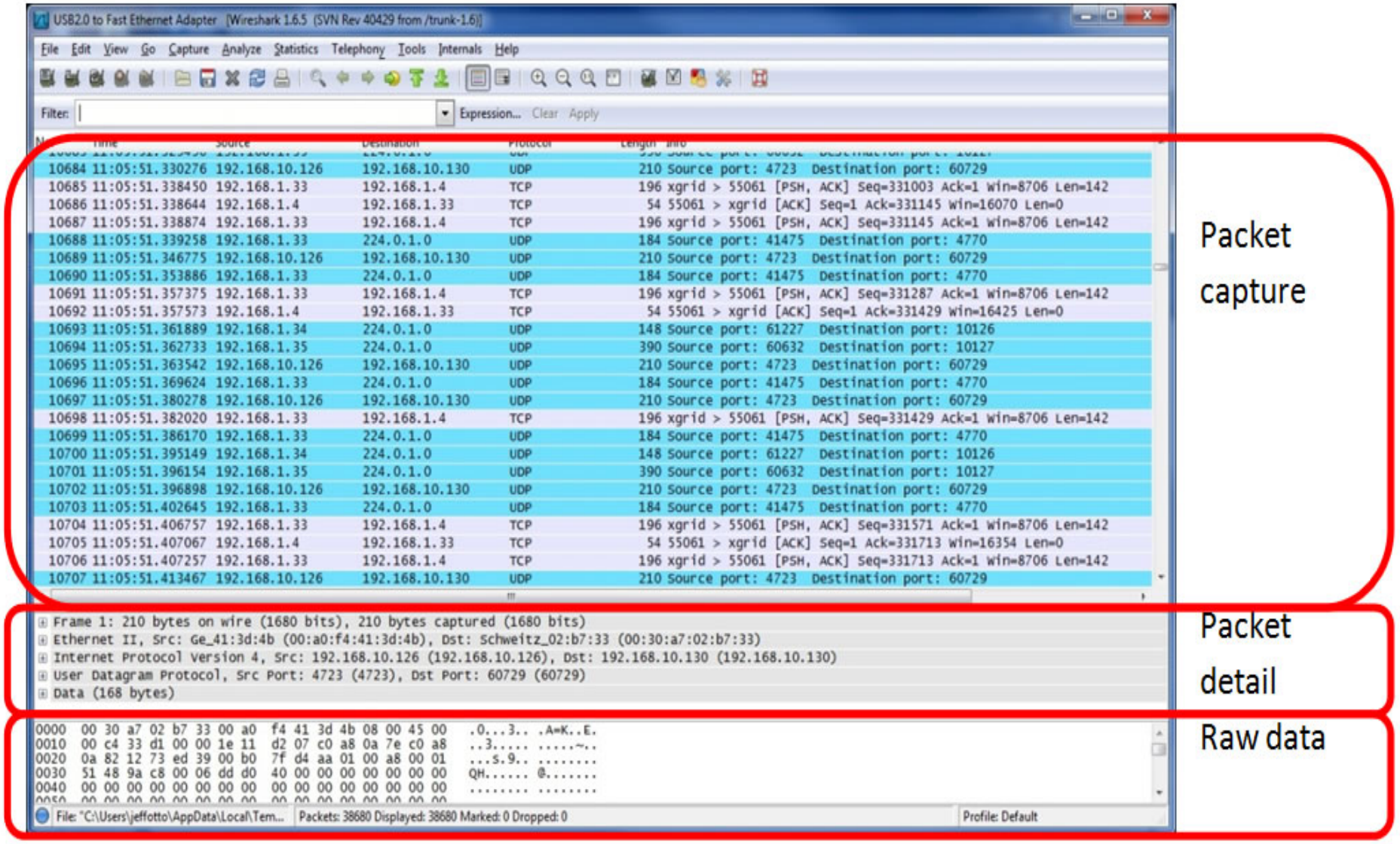
Wiki: https://en.wikipedia.org/wiki/Internet_protocol_suite

Traffic on TCP/IP Protocol Stack





WireShark: Traffic Capture for Analysis



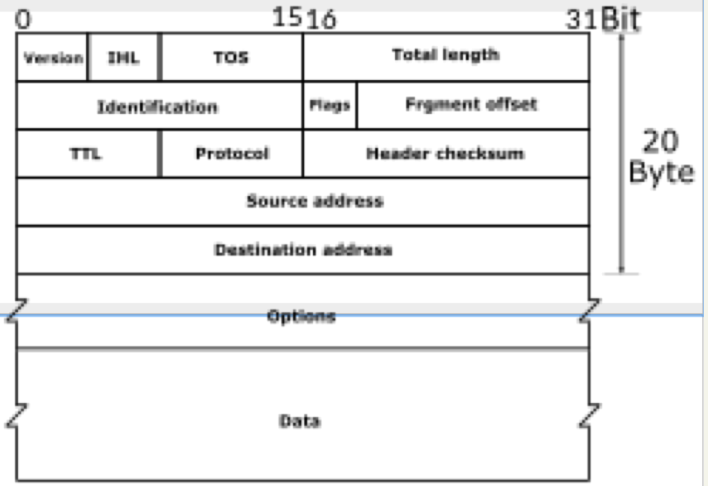
Traffic Analysis for Smurf Attack

Apply a display filter ... <⌘/> Expression... + Apply this filter

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=1/256, ttl=64 (reply in 2)
2	0.001342	10.10.1.50	10.10.1.1	ICMP	98	Echo (ping) reply id=0x6658, seq=1/256, ttl=255 (request in 1)
3	1.001028	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=2/512, ttl=64 (reply in 4)
4	1.031570	10.10.1.50	10.10.1.1	ICMP	98	Echo (ping) reply id=0x6658, seq=2/512, ttl=255 (request in 3)
5	2.002809	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=3/768, ttl=64 (reply in 6)
6	2.004017	10.10.1.50	10.10.1.1	ICMP	98	Echo (ping) reply id=0x6658, seq=3/768, ttl=255 (request in 5)
7	3.004298	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=4/1024, ttl=64 (reply in 8)
8	3.004507	10.10.1.50	10.10.1.1	ICMP	98	Echo (ping) reply id=0x6658, seq=4/1024, ttl=255 (request in 7)
9	4.003576	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=5/1280, ttl=64 (reply in 10)
10	4.004785	10.10.1.50	10.10.1.1	ICMP	98	Echo (ping) reply id=0x6658, seq=5/1280, ttl=255 (request in 9)
11	5.006017	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=6/1536, ttl=64 (reply in 12)
12	5.007230	10.10.1.50	10.10.1.1	ICMP	98	Echo (ping) reply id=0x6658, seq=6/1536, ttl=255 (request in 11)
13	6.006544	10.10.1.1	10.10.1.50	ICMP	98	Echo (ping) request id=0x6658, seq=7/1792, ttl=64 (reply in 14)

```
▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▼ Ethernet II, Src: Intel_b7:20:46 (00:04:23:b7:20:46), Dst: Intel_b7:23:a0 (00:04:23:b7:23:a0)
  ▶ Destination: Intel_b7:23:a0 (00:04:23:b7:23:a0)
  ▶ Source: Intel_b7:20:46 (00:04:23:b7:20:46)
    Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 10.10.1.1, Dst: 10.10.1.50
▼ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x9b87 [correct]
    [Checksum Status: Good]
    Identifier (BE): 26200 (0x6658)
    Identifier (LE): 22630 (0x5866)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
    Timestamp from icmp data: Apr 25, 2018 20:29:19.000000000 PDT
    [Timestamp from icmp data (relative): 0.698957000 seconds]
```

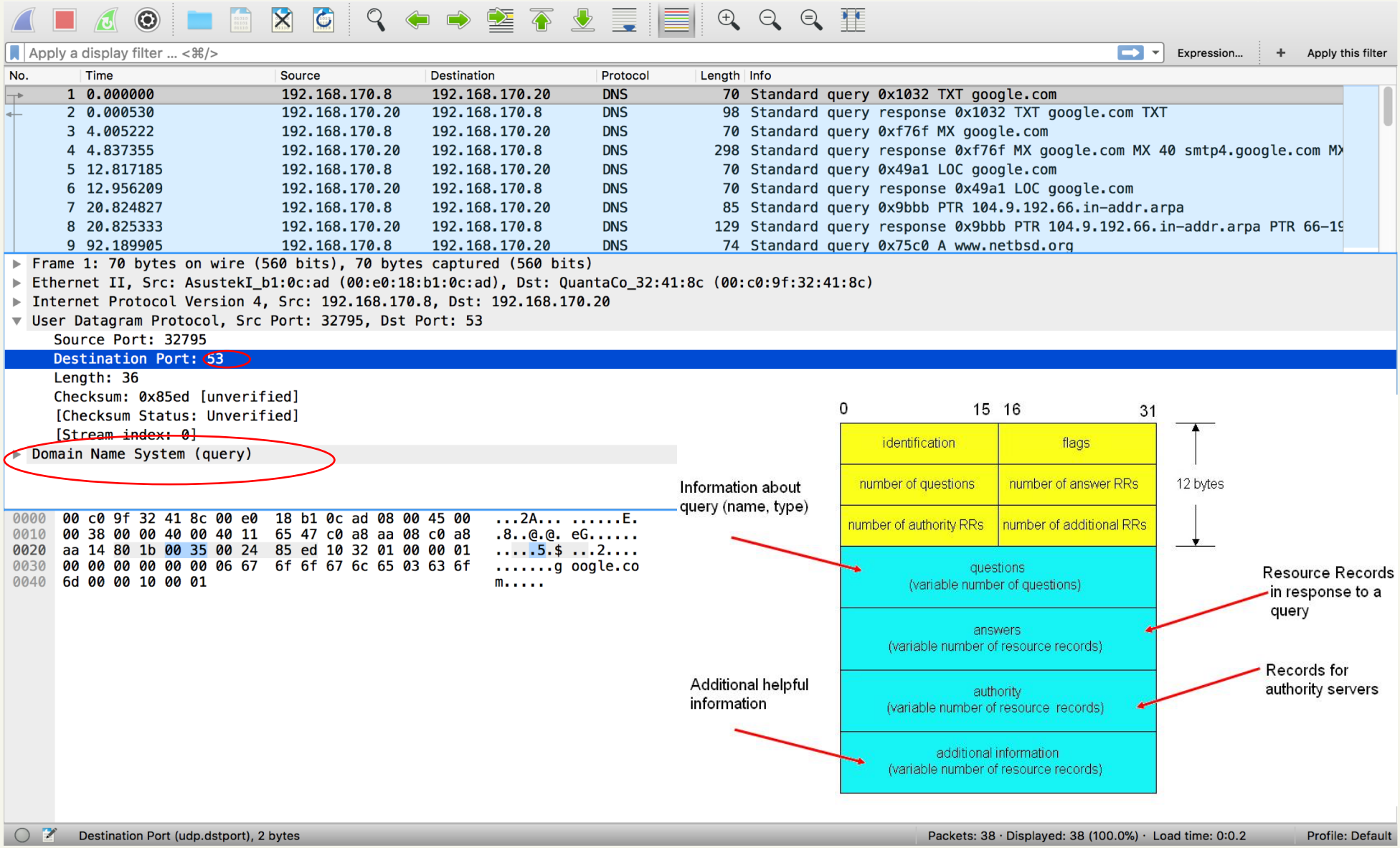
```
0000  00 04 23 b7 23 a0 00 04  23 b7 20 46 08 00 45 00   ..#.#... #. F..E.
0010  00 54 e1 ef 40 00 40 01  42 73 0a 0a 01 01 0a 0a   .T..@.@. Bs......
0020  01 32 08 00 9b 87 66 58  00 01 8f 47 e1 5a 00 00   .2....fX ...G.Z..
0030  00 00 bc a9 0a 00 00 00  00 00 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```



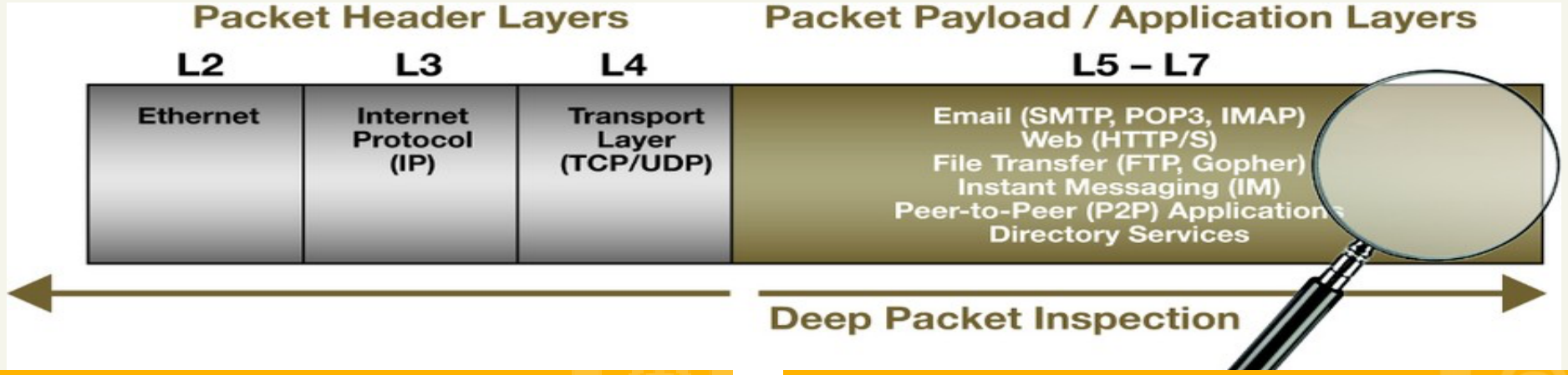
Type (eth.type), 2 bytes — Packets: 24 · Displayed: 24 (100.0%) · Load time: 0:0.2 — Profile: Default

https://en.wikipedia.org/wiki/IPv4

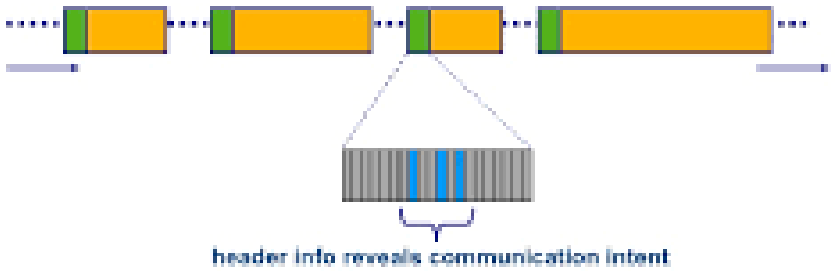
Traffic Analysis for DNS Attack



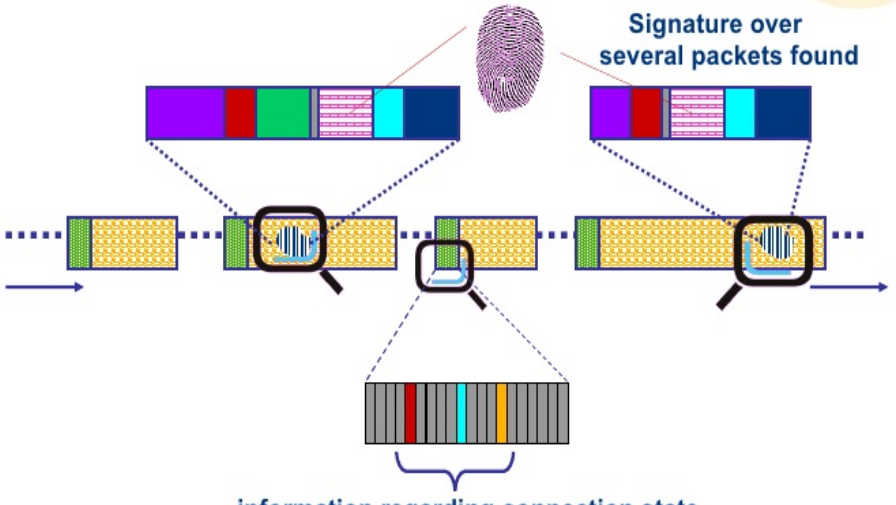
Traffic Analysis for DPI



Shallow (Standard) Packet Inspection



Deep Packet Inspection



Examples for Traffic Analysis

Normal?	Abnormal?	Example
A server accepts requests from clients	Over 100,000 SYN requests per less than second and lasts over 5 minutes	TCP SYN Flooding
A client connects to few destination hosts/ports	Over 1000 connection requests per second to destination hosts/ports	Port Scan or IP Scan
The source address is NOT the same as the destination address	The src address is the same as the dest addr.	LAND attack
The traffic rate for this network scope is usually around several Gbps	Over 10K Gpbs traffic rate appears in a small LAN	Zero-day attack (flooding attacks)
Various packet sizes	Fixed packet size (UDP/1434, packet size = 404)	SQL Slammer

Other Examples

- Snort (Signature-based NIDS)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139
flow:to_server,established
content:"|eb2f 5feb 4a5e 89fb 893e 89f2|"
msg:"EXPLOIT x86 linux samba overflow"
reference:bugtraq,1816
reference:cve,CVE-1999-0811
classtype:attempted-admin
```

Wiki: https://en.wikipedia.org/wiki/Snort_(software)

- Bro (now Zeek) (Abnormal-based NIDS)

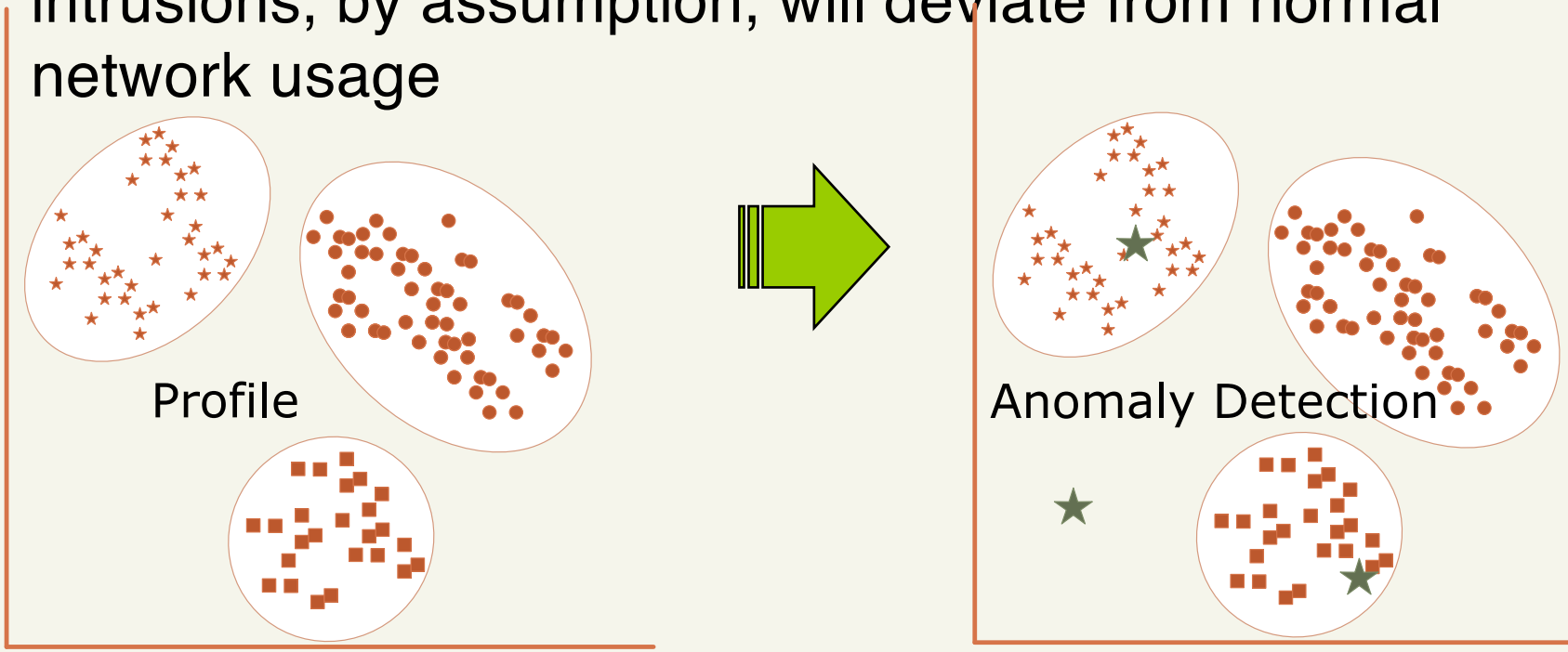
```
http_request(1.2.3.4/4321→5.6.7.8/80, "GET", "/index.html")
```

```
event http_request(c: connection, method: string, path: string)
{
    if ( method == "GET" && path == "/etc/passwd" )
        NOTICE(SensitiveURL, c, path);
}
```

Wiki: https://en.wikipedia.org/wiki/Zeek

Machine Learning & IDS

- One popular strategy for finding attacks is
  - monitoring a network's activity for *anomalies*: deviations from profiles of normality previously learned from benign traffic
  - detect new types of intrusions because these new intrusions, by assumption, will deviate from normal network usage



Machine Learning & IDS

Challenges

- **Differentiating anomalous activity from normal network traffic** is complicate and tedious.
- It is hard for a human analyst to search through vast amounts of data for anomalous sequences of network activities.

Machine-learning techniques

- reduce the human effort required to build these systems
- improve performance in finding patterns and differentiating abnormal behaviors.

Algorithms/Models

- train a model with reference input to “learn” its patterns
- use the model for the actual detection process

Machine Learning for Anomaly Detection

One popular strategy for finding attacks

anomalies - deviations from profiles of normality learned from normal traffic

Assumption – intrusions or attacks will deviate from normal network usage patterns

Machine Learning & IDS

- Machine-learning sees in many other areas of computer science, where it often results in large-scale deployments in the commercial world
  - product recommendations systems such as used by Amazon and Netflix
  - optical character recognition systems
  - natural language translation (e.g. Amazon Alexa, Google home)
  - spam detection
  - Optimizing web search
  - Biosecurity

Machine Learning & IDS

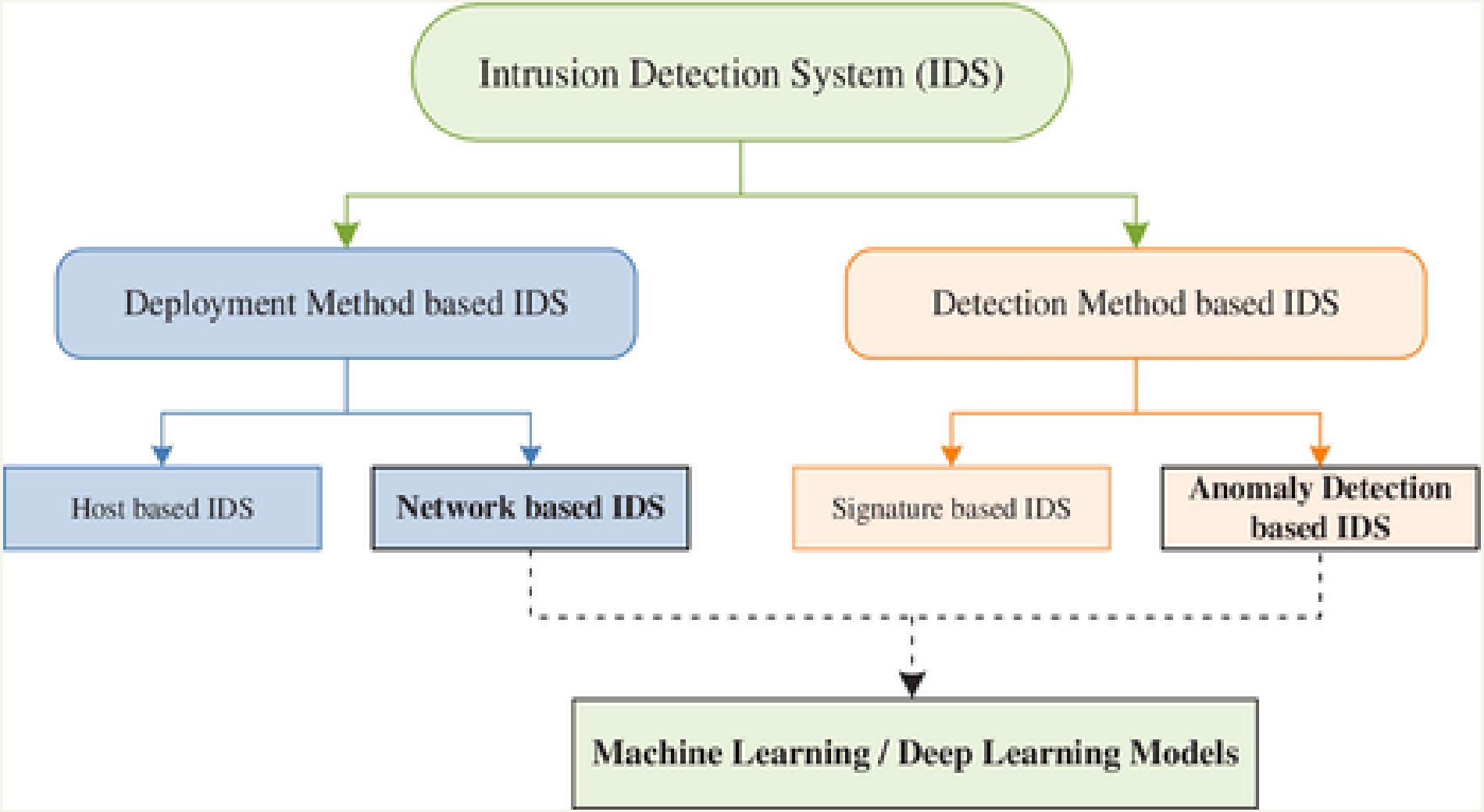
Table 1
Comparisons of related work.

Work	Technique	Dataset	Problem domain	Evaluation method	Baseline	Feature selection
Eesa et al. [6]	DT[a]	KDD-Cup 99	Anomaly detection	DR[b], FP[c], accuracy, ROC curve[d]	N/A	Yes
de la Hoz et al. [4]	GHSOM[e]	KDD-Cup 99 + DARPA 1999	Anomaly detection	DR, ROC curve	NB[f], RF[g], DT, AdaBoost	Yes
Feng et al. [5]	SVM[h]+ant colony network	KDD-Cup 99	Anomaly detection	DR, FP, FN[i]	SVM	No
Kim et al. [17]	DT + SVM	A modified version of KDD-Cup 99	Anomaly and misuse detection	DR, ROC curve	DT, SVM	No
Baig et al. [2]	GMDH[j]	KDD-Cup 99	Anomaly detection	Accuracy, Recall, Precision, FP, FN, ROC	NB, ANN[k]	Yes
Shin et al. [29]	Markov chain + probabilistic modeling of network events	DARPA 2000	Anomaly detection	DR, FP, ROC	Markov chain model	Yes
Lin et al. [19]	SA[l]+DT, SVM	KDD-Cup 99	Anomaly detection	DR	DT, SVM	Yes
Sangkatsanee et al. [27]	DT, ANN, Ripper rule	Reliability Lab Data 2009/KDD-Cup 99	Anomaly detection	DR	N/A	Yes
Wang et al. [36]	Fuzzy clustering + ANN	KDD-Cup 99	Anomaly detection	Precision, Recall, F-measure	DT/NB[m], ANN	No
Tajbakhsh et al. [32]	FL[n]+AR[o]	KDD-Cup 99	Anomaly detection	DR, FP	SVM, *k*-NN	No
Tong et al. [33]	RBF, Elman neural networks	DARPA 1999	Anomaly detection	DR, FP	ANN, SOM[p]	No
Giacinto et al. *[8]*	*k*-means, SVM ensembles	KDD-Cup 99	Anomaly detection	DR	N/A	No
Hu et al. [12]	AdaBoost DT	KDD-Cup 99	Anomaly detection	DR, FA[q], Run time	N/A	No
Xiang et al. [37]	Bayes clustering + DT	KDD-Cup 99	Anomaly detection	DR, Run time	DT	Yes
Abadeh et al. [1]	GA[r]+FL	DARPA 1998	Anomaly detection	DR, FA	FL	No
Chen et al. [3]	GA + ANN	DARPA 1998	Anomaly detection	FP, FN	ANN, 2 layer ANN	Yes
Hansen et al. [11]	GA	KDD-Cup 99	Anomaly detection	DR	N/A	No
Khan et al. [16]	SOM + SVM	DARPA 1998	Anomaly detection	FP, FN, accuracy	SVM	No
Li and Guo [18]	TCM *k*-NN	KDD-Cup 99	Anomaly detection	TP[s], FP	SVM, ANN, *k*-NN	No
Liu et al. [20]	SOM + ANN	DARPA 1998	Anomaly and misuse detection	DR, FA, FP	SVM, DT, SOM	Yes
Ozyer et al. [25]	GA + FL	KDD-Cup 99'	Anomaly and misuse detection	DR	GA	No
Peddabachigari et al. [26]	DT + SVM	KDD-Cup 99'	Anomaly and misuse detection	Accuracy	SVM, DT	No
Shon and Moon [30]	GA + SVM	DARPA 1999	Anomaly detection	DR, FP, FN	SVM	Yes
Shon et al. [31]	GA + ANN/*k*-NN/SVM	DARPA 1998	Anomaly detection	DR, FP, FN	ANN, *k*-NN, SVM	Yes
Sarasamma et al. [28]	Hierarchical SOM	KDD-Cup 99	Anomaly detection	DR. FP	SOM	Yes
Zhang et al. [38]	C-means clustering + ANN	KDD-Cup 99'	Anomaly and misuse detection	DR, FP	ANN	No

Machine Learning & IDS

- Techniques – Various machine-learning methods
  - DT: Decision Tree References
  - SVM: support vector machine
  - ANN: Artificial Neural Networks
  - SOM: self-organizing maps, etc.
  - Others
- Dataset for network intrusion
  - KDD99, DARAPA1998, 1999, etc.
- Problem Domain
  - Anomaly Detection (most)
  - Misuse Detection

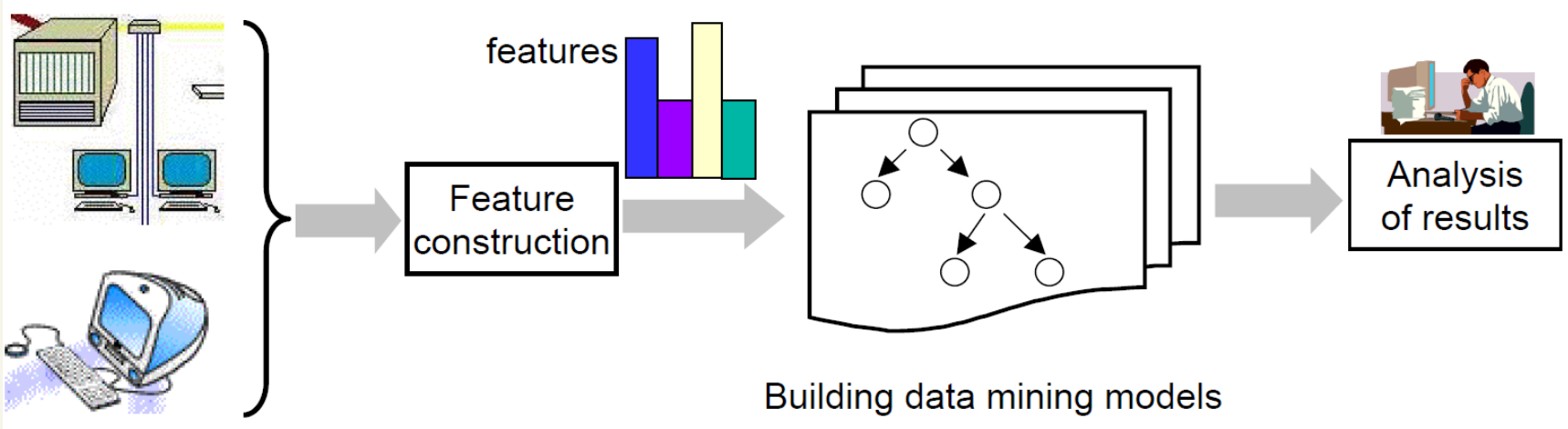
Overview of IDS



Source: https://onlinelibrary.wiley.com/doi/full/10.1002/ett.4150

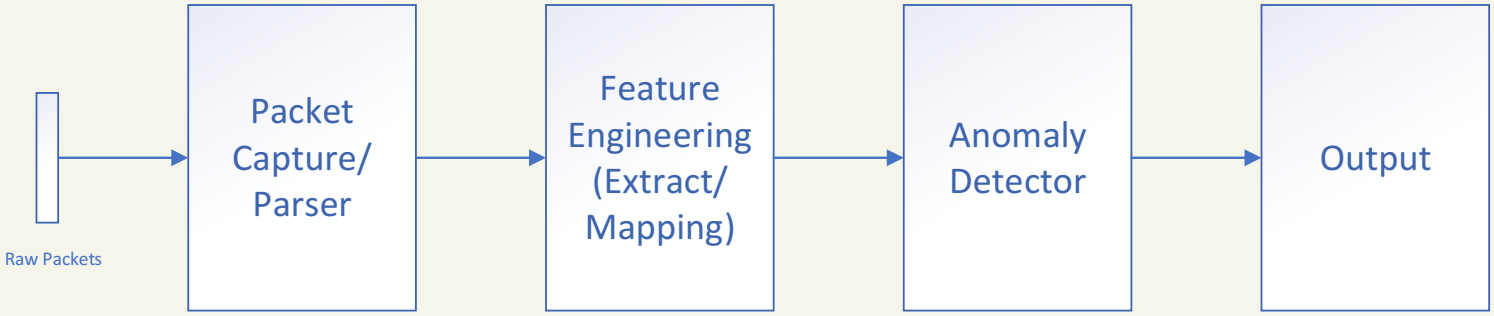
Machine Learning & IDS

- Converting Raw Data into Features
  - Features will be used in detection models
- Building Intrusion Detection Model
  - Misuse Detection Models
  - Anomaly Detection Models
- Analysis and Summarization of Results



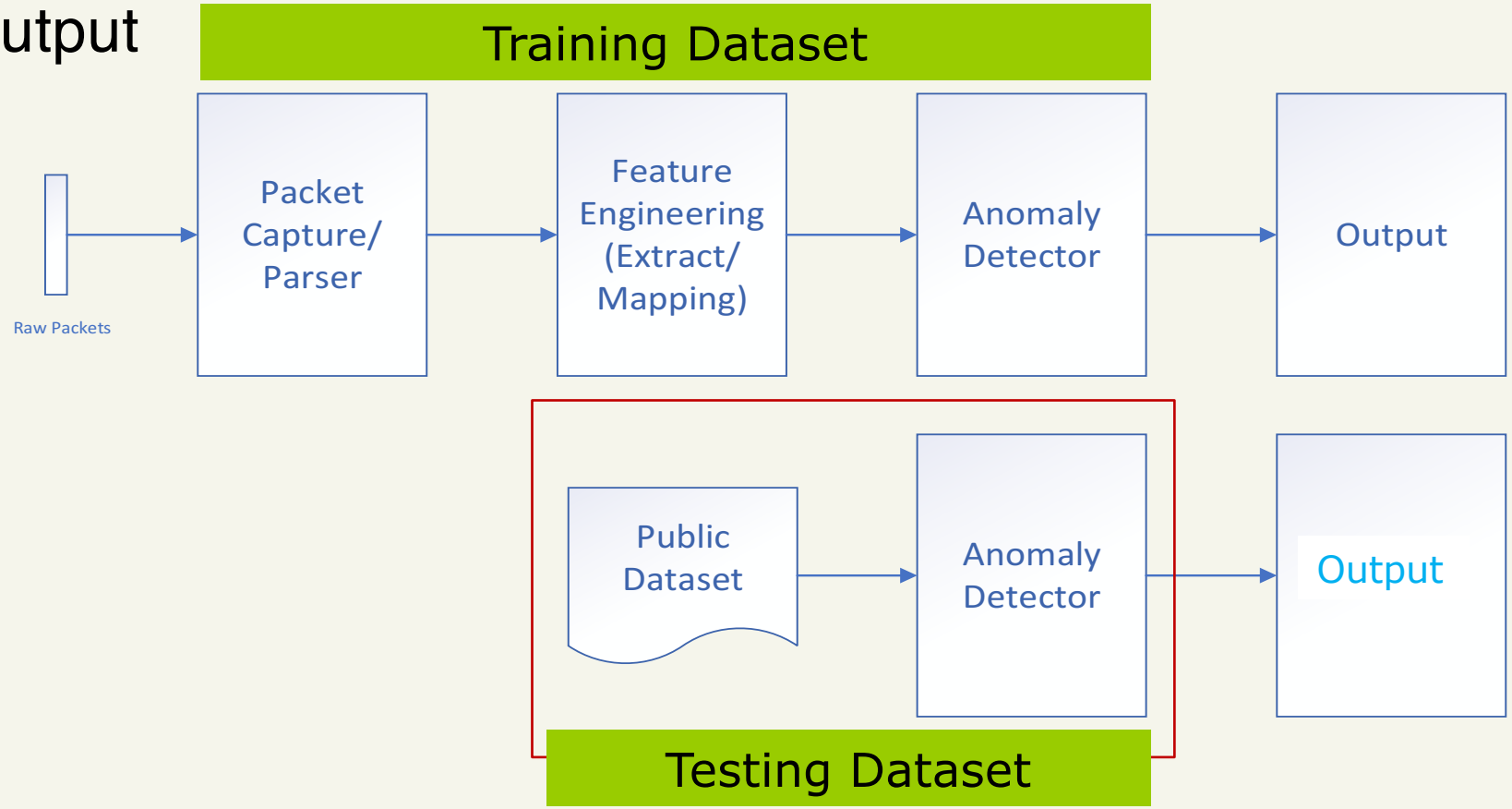
Machine Learning & IDS

- Packet Capture/Parser
- Feature Extract/Mapping
- Anomaly Detector
- Output



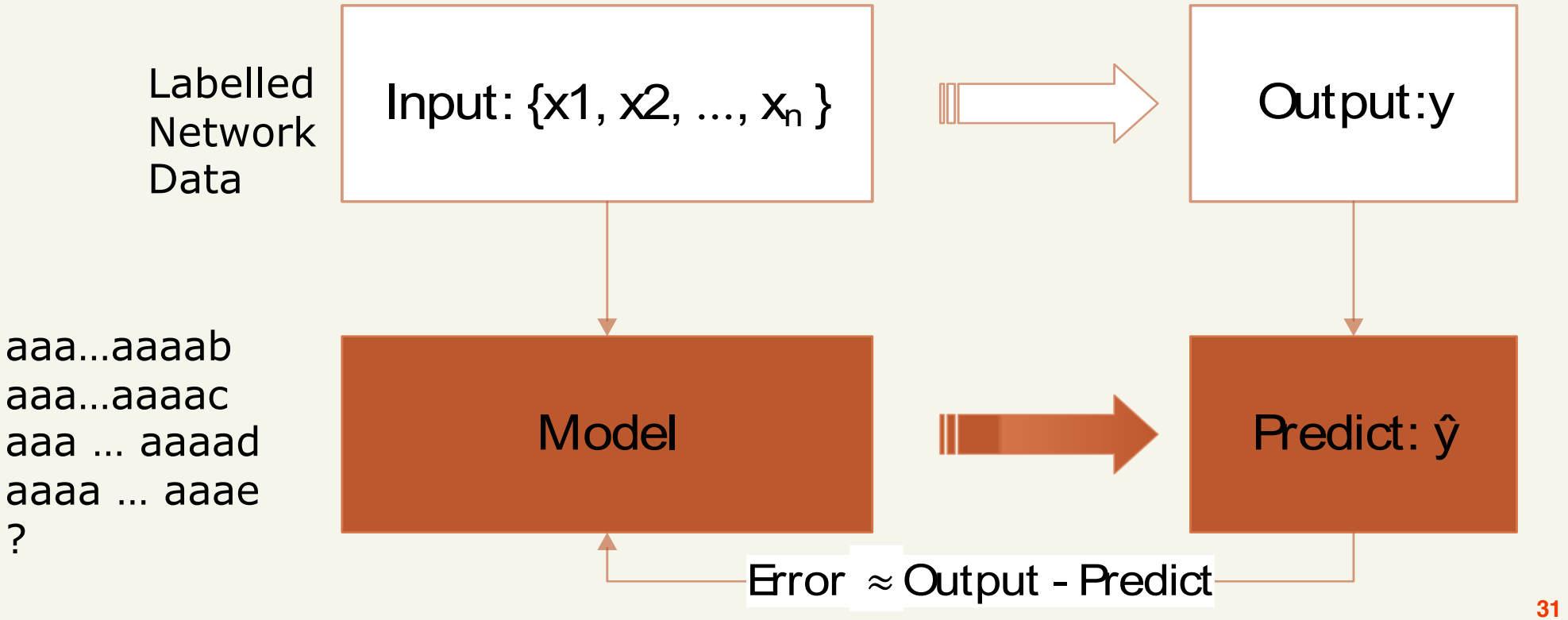
Machine Learning & IDS

- Packet Capture/Parser
- Feature Extract/Mapping
- Anomaly Detector
- Output



Supervised Learning

- Using machine learning algorithms to train on labelled network data, i.e., with instances pre-classified as being an attack or not.

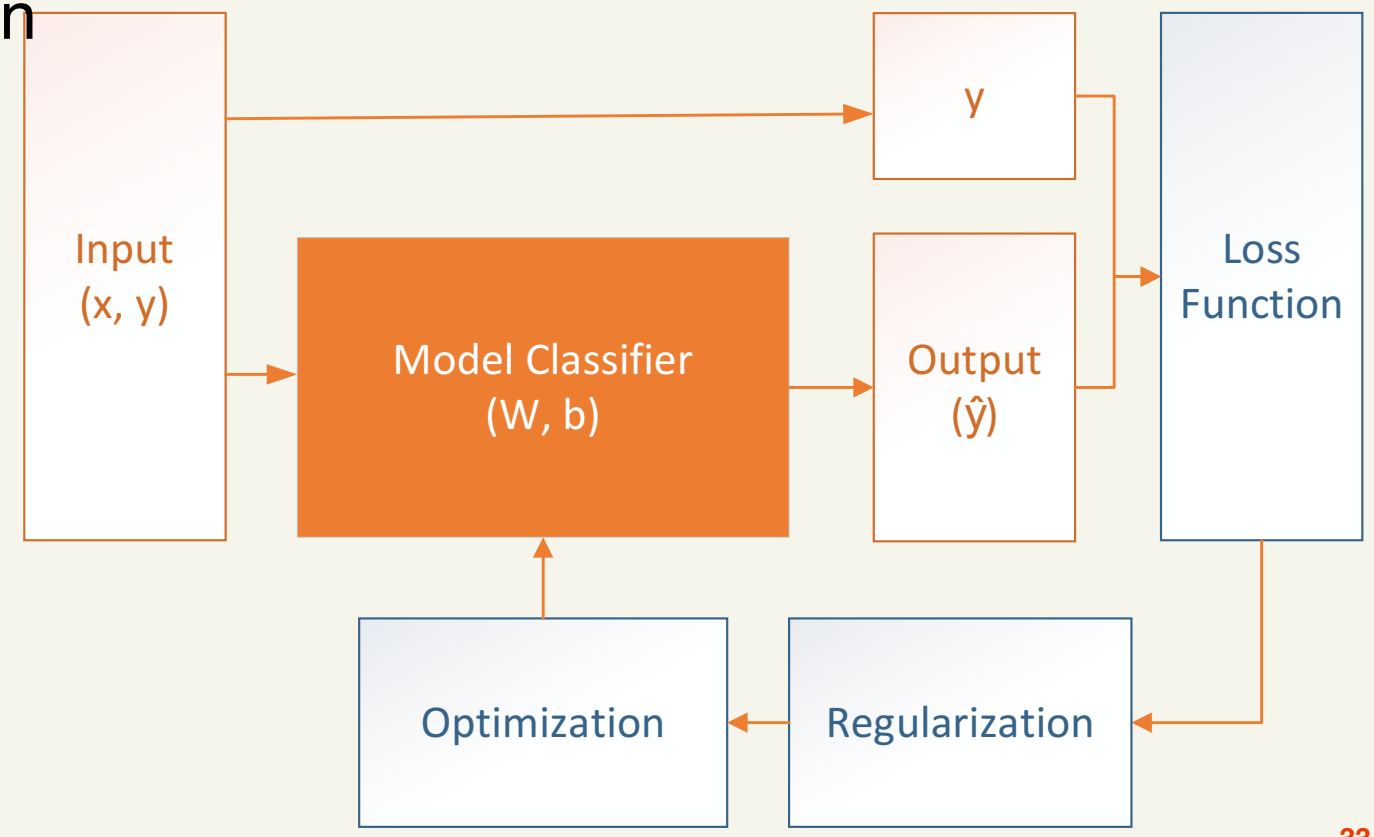


Classification for Anomaly Detection

- Intrusion (anomaly) detection can be approached as a classification problem that classifies audit events as belonging to the benign or malicious class.
- These algorithms normally output "classifiers", for example, in the form of decision trees or rules.
- An ideal application in intrusion detection would
  - gather sufficient normal and abnormal audit data for a user or a program
  - to correctly train the classifier and then
  - apply a classification algorithm to make decisions for the (future) audit data as belonging to the normal or abnormal class.

Anomaly Classification w/ Machine Learning

- Anomaly Classification
  - Machine Learning Model
  - Loss Function
  - Regularization
  - Optimization



Dataset for NIDS

- **Packet-level data**
  - The network packets received and transmitted can be captured by a specific application programming interface (API) called pcap
  - e.g. Libpcap, tcpdump, Wireshark, Snort, and Nmap
  - The features of the data vary with respect to the protocol that packet carries.
- **NetFlow Data**
  - introduced as a router feature by Cisco
  - The router or switch has the ability to collect IP network traffic as it enters and exits the interface.
  - Cisco's NetFlow version 5 defines a network flow as a unidirectional sequence of packets that share the exact same seven packet attributes:
    - ingress interface, source IP address, destination IP address, IP protocol, source port, destination port, and IP type of service.
- **But now more than this features**

Dataset for NIDS

- Public Dataset
  - DARPA 1998, 1999
  - KDD 99 :http://kdd.ics.uci.edu/databases/kddcup99/task.html
    - https://datahub.io/machine-learning/kddcup99
  - Kyoto 2006+, others
- URLs for Dataset
  - Winner: https://www.unb.ca/cic/datasets/index.html
  - https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data
  - http://kdd.ics.uci.edu/databases/kddcup99/
  - https://archive-beta.ics.uci.edu/ml/datasets/130
  - https://www.caida.org/data/passive/passive%202016%20dataset
  - https://github.com/shramos/Awesome-Cybersecurity-Datasets#honeypots
  - My dataset: https://drive.google.com/drive/u/1/folders/1Ykuo0XCUMY_veF526vwQaq8JjuhubO2P

KDD 99 Dataset

- The KDD 1999 data set [20], which was created for the KDD Cup challenge in 1999.
- The data set is based on DARPA 1998 TCP/IP data and has basic features captured **by pcap.**
- Additional features were derived by analyzing the data with time and sequence windows.
- The data set has three components—basic, content, and traffic features—making for **a total of 41 attributes.**

Basic Features of KDD99 Dataset

- 14 Basic Features

Duration	the length (seconds) of the connection
Service	the connection's service type, e.g., http, telnet
Source bytes	the number of data bytes sent by the source IP address
Destination bytes	the number of data bytes sent by the destination IP address
Count	the number of connections whose source IP address and destination IP address are the same to those of the current connection in the past two seconds
Same srv rate	% of connections to the same service in Count feature
Flag	the state of the connection at the time the connection was written

Basic Features of KDD99 Dataset

- 14 Basic Features

Dst host count	among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose source IP address is also the same to that of the current connection
Dst host srv count	among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose service type is also the same to that of the current connection
Dst host same src port rate	% of connections whose source port is the same to that of the current connection in Dst host count feature

Basic Features of KDD99 Dataset

- 14 Basic Features

Serror rate	% of connections that have “SYN” errors in Count feature
Srv serror rate	% of connections that have “SYN” errors in Srv count (the number of connections whose service type is the same to that of the current connection in the past two seconds) feature
Dst host serror rate	% of connections that have “SYN” errors in Dst host count feature
Dst host srv serror rate	% of connections that “SYN” errors in Dst host srv count feature

Four Main Categories of Attacks

- **NSL-KDD has 37 types of attack in 4 Categories**
- **DoS, denial of service**
  - For example, Ping-of-Death, Teardrop, smurf, SYN flood, and so on;
- **R2L (Remote to Local)**
  - Unauthorized access from a remote machine,
  - For example, guessing password;
- **U2R (User to Root)**
  - unauthorized access to local superuser privileges by a local unprivileged user,
  - For example, various buffer overflow attacks; and
- **PROBING,**
  - surveillance and probing,
  - for example, Port-Scan, Ping-Sweep.

Example of KDD99 Dataset

- Each dataset is one line of CSV-formatted data, containing 41 features, like this:

0,tcp,http,SF,215,45076,
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,
0.00,0.00,0.00,0.00,1.00,0.00,0.00,0,0,0.00,
0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal.

Raw Data -> Encoding -> Feature Selection

Kyoto 2006+ Dataset

- Attack data - Honeypots
  - Kyoto 2006+ datasets have built on the real traffic data which are obtained from diverse types of honeypots
  - All traffic data to/from honeypots have been collected, and thoroughly inspected using three security software programs to identify what happened on the networks.
- Normal traffic data
  - A server has been deployed on the same network with the honeypots.
  - The server has two main functions of mailing service and DNS server which services only one domain.
  - All traffic data related to the server are regarded as normal data.

Kyoto 2006+ Dataset

- The Kyoto 2006+ dataset consist twenty-four statistical features; fourteen features based on features of KDD Cup 99 dataset and ten additional features.
  - Among the original 41 features of the KDD Cup 99 dataset, insignificant features and content features were excluded in Kyoto 2006+ dataset, because they are not suitable for network-based intrusion detection systems.
  - Fourteen statistical features, which are significant and essential features, were extracted from honeypot data.
  - Addition to the 14 statistical features, additional 10 features were extracted, which allow investigating more effectively what kinds of attacks happened on networks.
- Reading research paper.

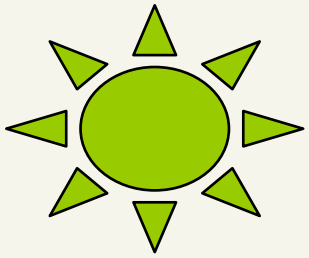
Kyoto 2006+ Dataset

- Label feature indicates whether the session was attacked or not
- "Unknown" attacks
  - Among traffic data captured from honeypots, there are some sessions that did not trigger any alerts but contained shellcodes.
- The summary of datasets for recent three years

	Year 2013	Year 2014	Year 2015
Total	93,934,707	110,533,528	136,719,395
Normal	6,450,396	5,092,822	6,581,188
Known Attack	87,418,261	105,420,614	130,135,437
Unknown Attack	66,050	20,092	2,770

Research Example: RICOCHET

- Improve accuracy
- Multiple classifiers are used for the identification of attacks and the extraction of the most uncertain network traffic subsets
- Reducing Ambiguous Data
  - Ambiguous data cannot be classified into attack or normal with 100% sure.

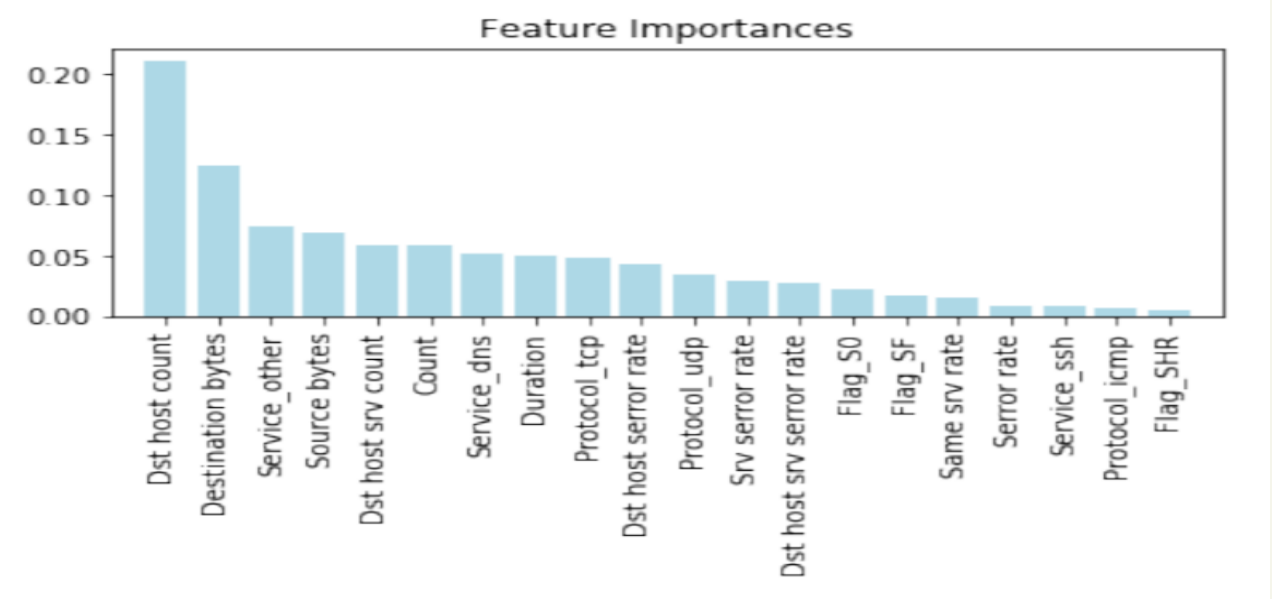


RICOCHET: Classification of Network Data

- Build classifiers with different machine learning techniques
  - Decision Tree,
  - Gradient Boosted Tree,
  - Random Forest and
  - Multi-layer Perceptron model
- Evaluate individual classifier using Receiver Operator Characteristic (ROC) graphs.
  - ROC graphs are useful tools for selecting classification models based on their performance with respect to the false positive and true positive rates.

RICOCHET: Feature Engineering

- In figure we plot the ranks of features in the Kyoto 2006+ dataset by their relative importance
  - the feature importance is normalized so that they sum up to 1.0.
  - First, 10 features among 47 features take 81% of importance.



	Full Dataset
1	Dst host count
2	Destination bytes
3	Service other
4	Source bytes
5	Dst host srv count
6	Count
7	Service dns
8	Duration
9	Protocol tcp
10	Dst host serror rate

RICOCHET: Analysis Kyoto 2006+ Dataset

- Comparison of 10 most important features
  - 50% of them are matched – “Features are local”

	Kyoto 2006+	KDD 99
1	Dst host count	duration
2	Destination bytes	protocol_type
3	Service other	service
4	Source bytes	Flag
5	Dst host srv count	src_bytes
6	Count	dst_bytes
7	Service dns	land
8	Duration	wrong_fragment
9	Protocol tcp	urgent
10	Dst host serror rate	hot

RICOCHET: Experiment Results

- A perfect classifier would fall into the top-left corner of the graph with a true positive rate of 1 and a false positive rate of 0.

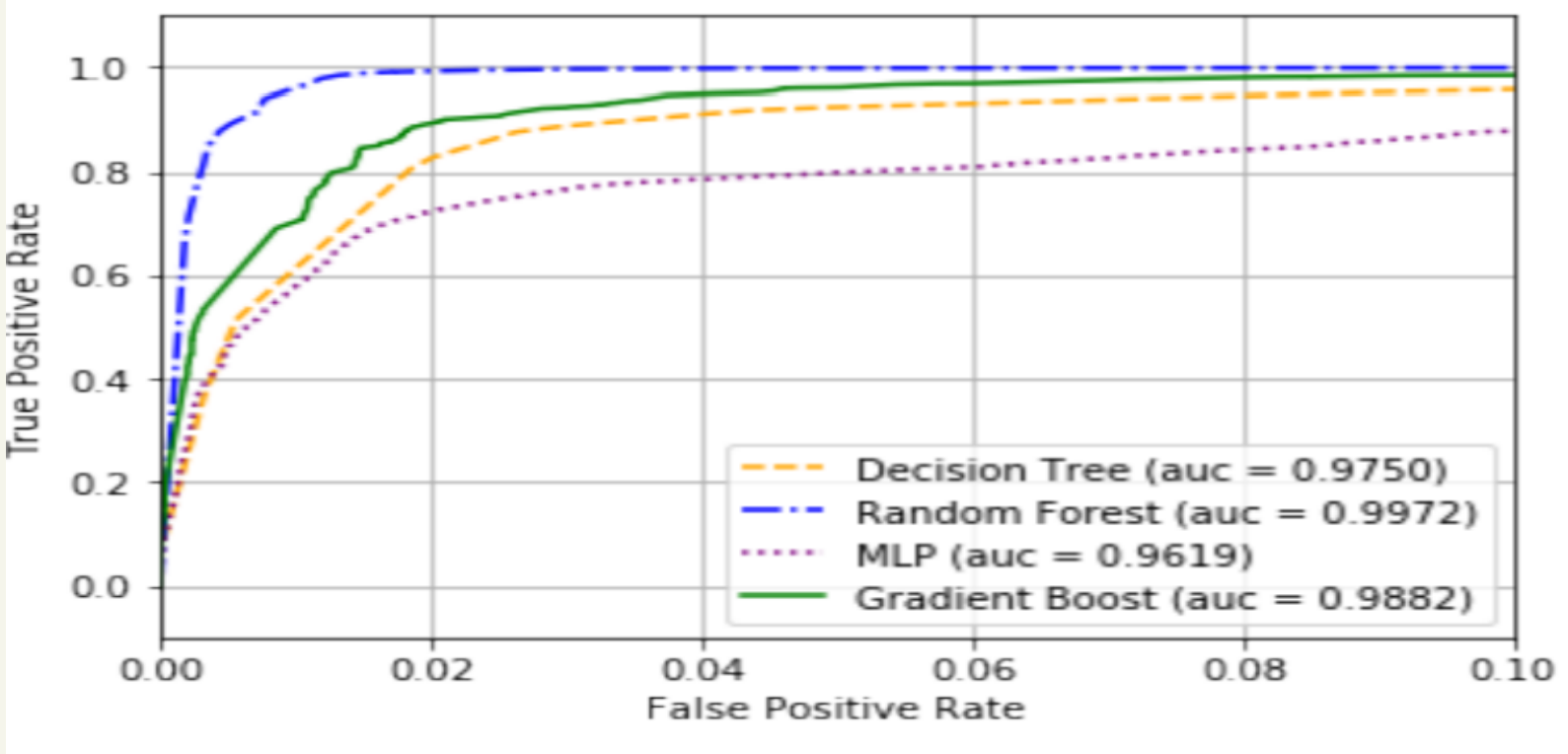


Figure 3: ROC-curves for classifiers evaluated

RICOCHET: Experiment Results

- Classify Confident and Ambiguous Dataset
- Performance of Intrusion detection of Confident and Ambiguous Dataset

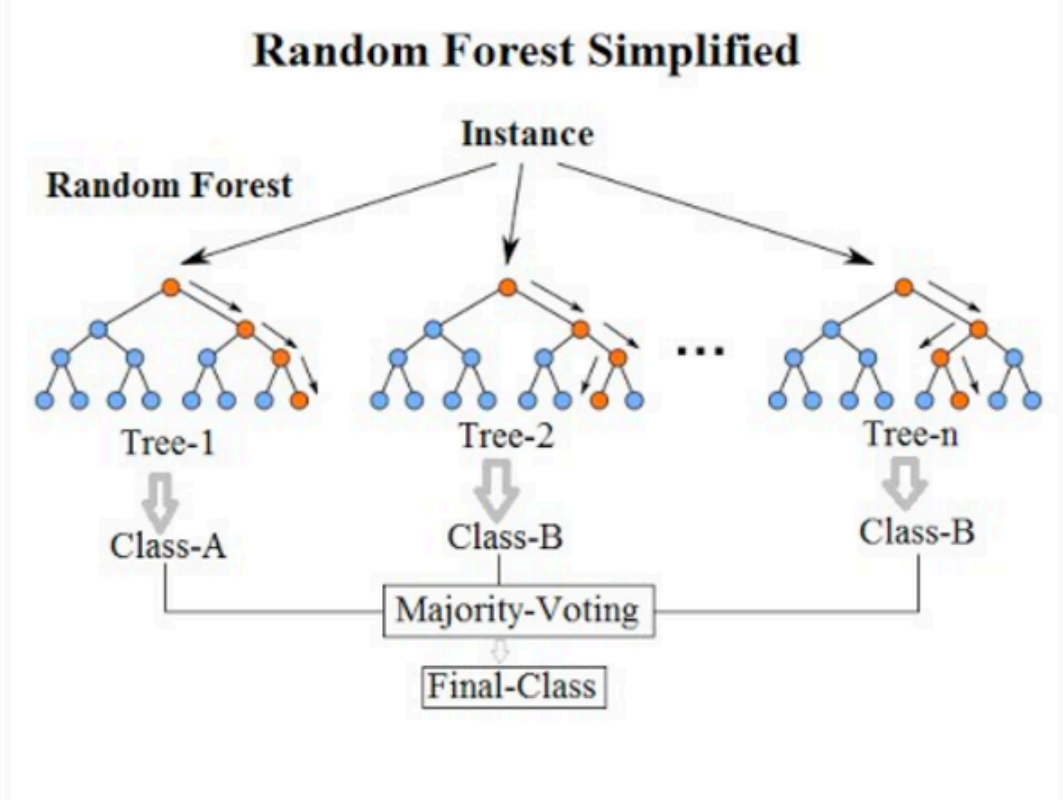
Dataset	ACC	PRE	REC	F1
Full	0.9665	0.9618	0.9711	0.9664
Confidence	0.9915	0.9958	0.9871	0.9914
Ambiguous	0.7477	0.7030	0.8267	0.7598

	ACC	PRE	REC	F1
Before	0.7477	0.7030	0.8267	0.7598
After	0.9029	0.9077	0.9194	0.9135

- Confusion Matrix: https://en.wikipedia.org/wiki/Confusion_matrix

Example 2: Random Forest w/ NSL KDD

- Random Forest
  - Refer to Wiki
  - Supervised learning
    - As part of their construction, random forest predictors naturally lead to a similarity measure among the observations.



- Demo: https://www.youtube.com/watch?v=-KgQLG7Q0s4&t=947s

Limitations in Supervised Model

- In most circumstances, labelled data is not readily available since it is time consuming and expensive to manually classify it.
- Purely normal data is also very hard to obtain in practice, since it is very hard to guarantee that there are no intrusions when we are collecting network traffic.
- What is considered to be normal depends on the local traffic observed by the NIDS.
- Attacks change overtime and new ones are constantly being discovered, so continuous maintainable of a malicious attack traffic repository may be impractical.

Supervised vs Unsupervised Approaches

- Supervised (classification) and unsupervised learning (clustering) techniques for detecting malicious activities.
- The supervised algorithms in general show better classification accuracy on the data with known attacks (the first scenario).
- However, if there are unseen attacks in the test data, then the detection rate of supervised methods decreases significantly. This is where the unsupervised techniques perform better as they do not show significant difference in accuracy for seen and unseen attacks.

Limitation of Anomaly Detection

- **High False Alerts Rates**
  - The high error rate is one of the primary reasons for the lack of success of machine learning-based intrusion detection system in operational settings.
- For anomaly detection aiming to find *novel* attacks, one cannot train on the attacks of interest, but *only* on normal traffic. Since ML requires
  - One needs to train a system with specimens of ***all* classes**
  - The number of representatives found in the training set for ***each* class should be large**
- Closed world assumption is not practical

"The idea of specifying only positive examples and adopting a standing assumption that the rest are negative"

Challenges in ML Approaches

- Intrusion detection domain exhibits characteristics that are not well aligned with the requirements of machine learning.
- These include (R. Sommer, V. Paxson. 2010):
  i. a very high cost of errors;
  ii. lack of training data;
  iii. a semantic gap between results and their operational interpretation;
  iv. enormous variability in input data; and
  v. fundamental difficulties for conducting sound evaluation.

  (cf. Oranges vs Lemons)

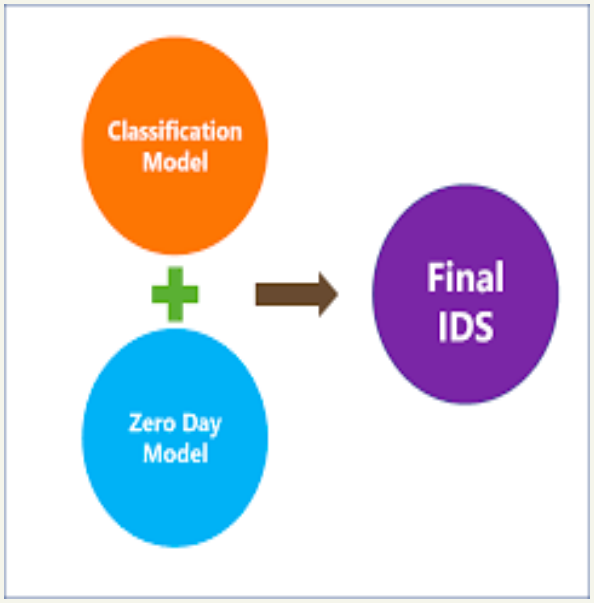
Challenges in ML Approaches

- **Feature Extraction and Mapping**
  - Network traffic cannot be used directly as input to current ML algorithms.
  - Instead input data must be formatted as fixed-size feature vectors.
  - The choice of features has a large impact on the detector's capability.
  - What features should be constructed?
  - Should that information be monitored over time?

- Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016, May). A deep learning approach for network intrusion detection system.

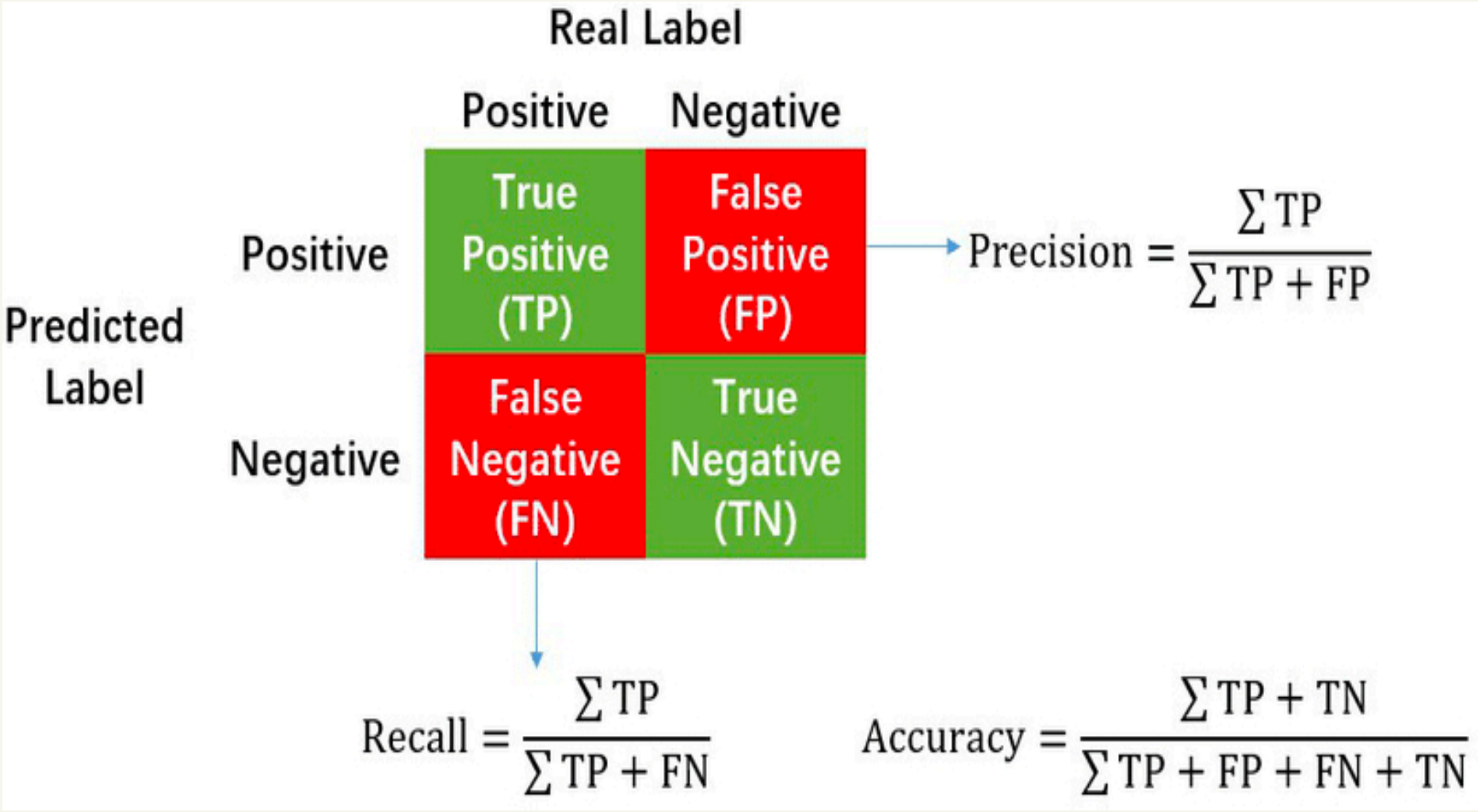
- Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection.

Q & A



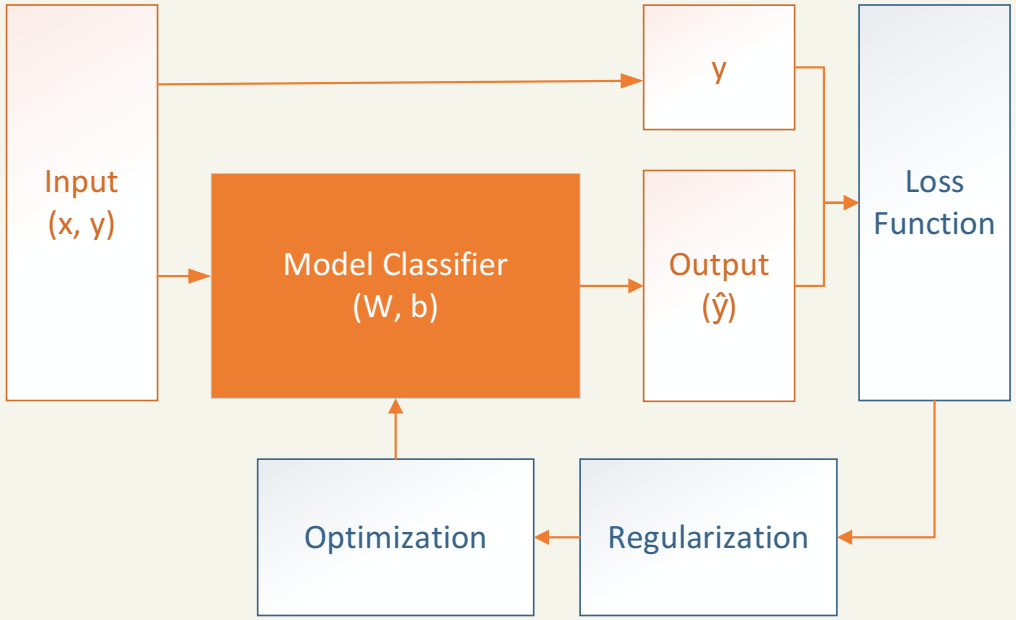
For ML & NIDS

Confusion Matrix for ML



Anomaly Classification w/ Machine Learning

- Anomaly Classification
  - Machine Learning Model
  - Loss Function
  - Regularization
  - Optimization



Input

- $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$,
  - $x^{(i)}$: feature vector (n-dimensional)

$$x^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right)$$

- $\mathrm{Y} = \{y^{(1)}, y^{(2)}, \cdots, y^{(m)}\}$
  - $y^{(i)}$: Expected Output (scalar)

Input: Pre-Processing

- Numeric features – Rescaling of the features to a range of $[0,1]$

$$x_{std}^{i} = \frac{x^{i} - \mu_x}{\sigma_x}$$

where μ_x - mean of a feature and σ_x - standard deviation

- Categorical features – one hot encoding
  - Binary values can then be used to indicate the value of a sample

Model: Logistic Regression

Model: a weighted sum of input features

$$z = W \cdot X + b = \sum_i w_i \cdot x_i + b$$

- $W = (w_1, w_2, \cdots, w_n)$ – weights
- b – bias

Output: Logistic Regression

- Output: the logistics of the weighted sum of the input features

$$\hat{p} = \sigma(W \cdot X + b)$$

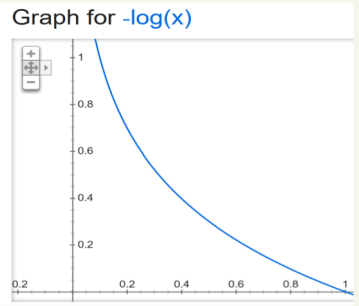
where σ is sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$

- Make Prediction:

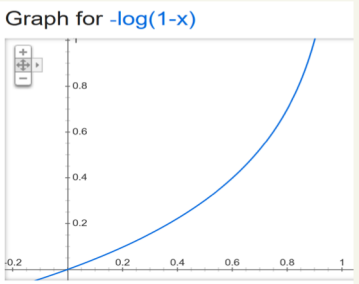
$$\hat{y} = \begin{cases} 0 & if \quad \hat{p} < 0.5 \ (attack \ class) \\ 1 & if \quad \hat{p} \geq 0.5 \ (benign \ class) \end{cases}$$

Lost Function

- Cross-Entropy Function



$$L(\theta) = \begin{cases} -\log(\hat{p})\ (low\ loss\ ) & if\ \ y = 1 \\ -\log(1-\hat{p})\ \ (high\ loss)\ if & y = 0 \end{cases}$$



- Lost function over the whole training set

$$L(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{p}^{(i)}) + (1-y^{(i)})\log(1-\hat{p}^{(i)})\right]$$

Regularization

- Regularizer - Minimize the generalization error
- L2-Regularization

$L(W, b)$

$$= \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{p}^{(i)}) + (1-y^{(i)})\log(1-\hat{p}^{(i)})\right] + \frac{\lambda}{2}W^{T}W$$

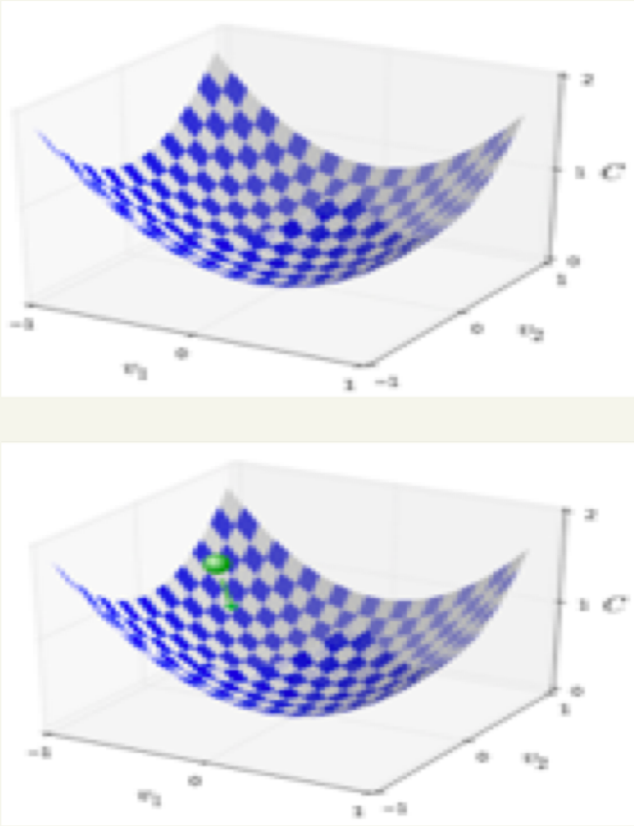
where λ is a hyperparameter

Optimization

- Gradient Descent
  - In the direction of decreasing gradient, update the parameters with learning rate η

$$W_j \leftarrow W_j - \eta \frac{\partial}{\partial W_j} L(W, b)$$

$$b \leftarrow b - \eta \frac{\partial}{\partial b} L(W, b)$$



Lab: Pre-Processing

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

num_attribs = list(kdd20_data_num)
cat_attribs = ["protocol_type","service", "flag"]

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('std_scaler', StandardScaler()),
    ])

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('cat_encoder', OneHotEncoder(n_values='auto')),
    ])
```

num_attribes: numeric features

cat_attribs: categorical features

StandardScaler - rescales numeric data in range [0, 1]

OneHotEncoder – categorical features

Lab: Select and Train Model

Binary Classification

normal - 1 abnormal - 0

```
kdd20_bi_labels = (kdd20_labels == 'normal').astype(np.int) # 1- normal 0- attack
```

A. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(kdd20_prepared, kdd20_bi_labels)
```

Evaluate Training Error

```
kdd20_predictions = log_reg.predict(kdd20_prepared)
```

Lab: Hyperparameters in LogisticRegression Classifier

- Penalty: Used to specify the norm used in the penalization (regression)
- C: Inverse of regularization strength (learning rate)
- Solver: Algorithm to use in the optimization problem
- Max_tier: Maximum number of iterations taken for the solvers to converge.

Lab: Grid Search

- Exhaustive search over specified parameter values for an estimator.

```
from sklearn.model_selection import GridSearchCV
param_grid = [
    {'penalty':['l2'], 'C':[0.5, 1.0, 2.0], 'max_iter':[100, 300, 500]},
    {'penalty':['l1'], 'C':[0.5, 1.0, 2.0], 'max_iter':[100, 300, 500]}
]

grid_search = GridSearchCV(log_reg, param_grid, cv=3, scoring="accuracy")
grid_search.fit(kdd20_prepared, kdd20_bi_labels)
```

Lab: Grid Search

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'penalty': ['l2'], 'C': [0.5, 1.0, 2.0], 'max_iter': [100, 300, 500]}, {'penalty': ['l1'], 'C': [
0.5, 1.0, 2.0], 'max_iter': [100, 300, 500]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

```
grid_search.best_score_
```

```
0.9740780437457822
```

```
grid_search.best_params_
```

```
{'C': 2.0, 'max_iter': 500, 'penalty': 'l1'}
```

Network Intrusion Detection as Classification

- NSL-KDD
- Binary Classification – Malicious or Benign?
- Logistic Regression
  - Accuracy 97.4%
- Other approaches
  - Decision Tree
  - Gradient Boosted Tree
  - Random Forest Multi-layer Perceptron model
  - And so on …

Softmax Linear Classification

- Classification with multiple classes (normal, DOS, Probe, R2L, and U2R.)
- Given an input data set x
  - Compute a score $s_k(x)$ for each class k: $s_k(x) = W_k^T \cdot x + b_k$ (Linear)
  - Estimate the probability of each class using softmax

function $\hat{p}_k(x) = \frac{e^{s_k(x)}}{\sum_{j=1}^{N} e^{s_j(x)}}$

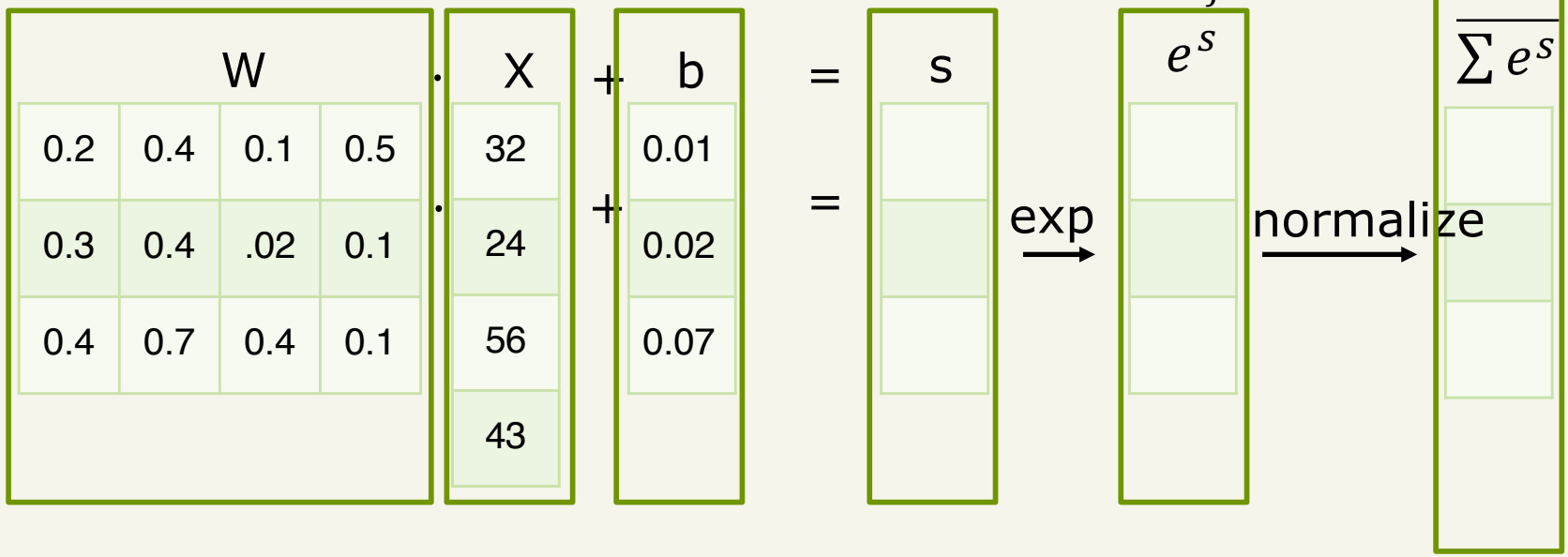
Predict the class with the highest estimated probability

$$\hat{k} = \arg\max_k \hat{p}_k$$

Softmax Linear Classification

$4\ features$
$3\ classes$

$$s_k(x) = W_k^T \cdot x + b_k \qquad \hat{p}_k(x) = \frac{e^{s_k(x)}}{\sum_{j=1}^{N} e^{s_j(x)}}$$



Lab: Code for Softmax Linear Classification

- Pre-processing data
  - Numeric features – standardize
  - Categorical features – Vectorize: One Hot Encoder

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import FeatureUnion

kdd20_data_num = kdd20_data.drop(["protocol_type","service","flag"], axis=1)

num_attribs = list(kdd20_data_num)
cat_attribs = ["protocol_type","service","flag"]

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('std_scaler', StandardScaler()),
    ])

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('cat_encoder', OneHotEncoder(n_values='auto')),
    ])

full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline)
    ])

kdd20_prepared = full_pipeline.fit_transform(kdd20_data)
```

Lab: Softmax Linear Classification

- **Label – 5 categories**
  - Normal – 0
  - DoS – 1
  - Probe – 2
  - U2R – 3
  - R2L – 4

```
kdd20_five_labels = kdd20_labels.replace([
'back',
'buffer_overflow',
'ftp_write',
'guess_passwd',
'imap',
'ipsweep',
'land',
'loadmodule',
'multihop',
'neptune',
'nmap',
'perl',
'phf',
'pod',
'portsweep',
'rootkit',
'satan',
'smurf',
'spy',
'teardrop',
'warezclient',
'warezmaster',
'normal'], [1,3,4,4,4,2,1,3,4,1,2,3,4,1,2,3,2,1,4,1,4,4,0])
```

Lab: Softmax Linear Classification

- Initialize Model
  - Dimension – 118
  - Class – 5
  - Initialize W and b
  - step_size
  - regularization strength

```
X = X_train.toarray()
y = y_train.values

D = 118 # dimensionality
K = 5 # number of classes

# initialize parameters randomly
W = 0.01 * np.random.randn(D,K)
b = np.zeros((1,K))

# some hyperparameters
step_size = 1e-0
reg = 1e-3 # regularization strength

# gradient descent loop
num_examples = X.shape[0]
```

Lab: Softmax Linear Classification

- Calculate
  - Scores , Probability, Data Loss, and Regularization Loss

```
# evaluate class scores, [N x K]
scores = np.dot(X, W) + b

# compute the class probabilities
exp_scores = np.exp(scores)
probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]

# compute the loss: average cross-entropy loss and regularization
correct_logprobs = -np.log(probs[range(num_examples),y])
data_loss = np.sum(correct_logprobs)/num_examples
reg_loss = 0.5*reg*np.sum(W*W)
loss = data_loss + reg_loss

if i % 100 == 0:
    print ('iteration %d: loss %f' % (i, loss))
```

Lab: **Softmax** Linear Classification

- Optimization
  - Calculate derivatives
  - Update parameters by Gradient Descent

```
# compute the gradient on scores
dscores = probs
dscores[range(num_examples),y] -= 1
dscores /= num_examples


# backpropate the gradient to the parameters (W,b)
dW = np.dot(X.T, dscores)
db = np.sum(dscores, axis=0, keepdims=True)

dW += reg*W # regularization gradient

# perform a parameter update
W += -step_size * dW
b += -step_size * db
```

Lab: Softmax Linear Classification

- Calculate test accuracy

```
# evaluate test set accuracy
X = X_test.toarray()
y = y_test.values

scores = np.dot(X, W) + b
predicted_class = np.argmax(scores, axis=1)
print ('test accuracy: %.4f' % (np.mean(predicted_class == y)))
```

- Accuracy is 97.3%

Summary

- Binary classification using Logistic Regression classifier
  - Accuracy = 97.4%
- Multi-classification using Softmax Linear Classification
  - Average Accuracy = 97.3%

Kicho's demo: https://www.youtube.com/watch?v=-KgQLG7Q0s4