IJBT (2018) 01: 23-39
RESEARCH ARTICLE

Machine learning in computer security

V. Rao Vemuri[1]

Abstract The past few years have witnessed a rise in the use of AI and Machine Learning techniques to a variety of application areas, such as image understanding and autonomous vehicle driving. Wireless and cloud technologies have also made it possible for millions of people to access and use services available via the internet. During the same period, the world has also witnessed a rise in cyber-crime, with criminals continually expanding their methods of attack. Weapons like ransomware, botnets, and attack vectors became popular forms of malware attacks. This paper examines the state-of-the-art in computer security and the use of machine learning techniques therein. True, machine learning did make an impact on some narrow application areas such as spam filtering and fraud detection. However – in spite of extensive academic research – it did not seem to make a visible impact on the problem of intrusion detection in real operational settings. A possible reason for this apparent failure is that computer security is inherently a difficult problem. Difficult because it is not just one problem; it is a group of problems characterized by a diversity of operational settings and a multitude of attack scenarios. This is one reason why machine learning has not yet found its niche in the cyber warfare armory. This paper first summarizes the state-of-the-art in computer security and then examines the process of applying machine learning to solve a sample problem.

Keywords: Machine Learning · Computer Security · Intrusion Detection

1 The status of cyber warfare

Cyber security, a subset of information security, is the practice of defending an organization's networks, computers and data from unauthorized digital access, attack or damage by implementing various processes, technologies and practices.

Network security, a subset of cyber security, aims to protect any data that is being sent through devices in a network to ensure that the information is not changed or intercepted. The role of network security is to provide protection from all types of cyber threats including viruses, worms, Trojan horses, zero-day attacks, hacker attacks, denial of service (DoS) attacks, and attacks by spyware, adware, ransomware, and so on.

✉ V. Rao Vemuri
rvemuri@ucdavis.edu

1 Professor, Department of Computer Science, University of California, Davis, USA

With the advent of wireless technology, cloud computing and Internet of Things (IoT), the world is becoming one huge integrated network. This means that an increasing amount of personal and corporate information exists in the cloud where it potentially interchanges with a multitude of devices. A compromised network does not only mean access to private banking details, but also access to public infrastructures such as traffic lights, GPS tracking systems, water services, and power plants. Under these circumstances, faster and more reliable intrusion detection techniques become necessary both to diagnose and prevent attacks. To meet this challenge, some are suggesting the use of the power of parallel computing platforms like MapReduce and Hadoop, an open-source software framework for distributed storage and processing of big data [7, 11].

1.1 Hacking has gone pro

At the turn of the century, almost all threats to computer systems were malware programs (viruses, worms, and Trojans) written by pranksters. Although some malware did harm, most simply annoyed people. Professional and state-sponsored hackers were there, but they were not the norm.

Nowadays almost all malware is created to steal money or corporate secrets [14]. Professional hackers make millions of dollars, victimizing individuals and corporations with almost no fear of being prosecuted. Malware has morphed from innocuous, funny viruses and worms to identity-stealing programs and ransomware. Advanced persistent threats (APTs), such as mobile surveillanceware like JadeRAT, officially or unofficially working on behalf of a foreign government, are the new normal. According to the FBI, cyberwar will turn black in the coming years with sinister activities like taking control of a moving vehicle from a distance or remotely turning off a heart pacemaker [1].

1.2 Breach detection tools have improved

Once antivirus scanners were the main tools for breach detection. Now, products have been developed to detect when someone is doing something malicious, even if that someone is a “legitimate” user.

Event monitoring systems are improving. Many companies are now storing and analyzing billions of events a day, using huge disk storage arrays. Intrusion detection has moved beyond detecting simple malicious activity to detecting anomalous events that are out of character for a company and its employees. Connections to known, questionable networks are tracked and reported like the antivirus detections of yesteryear. Data leak protection (DLP)[1] has become big business.

[1] Roger A. Grimes, “Make stolen data worthless”, http://www.infoworld.com/article/2969372/security/how-to-make-stolen-data-worthless.html, August 11, 2015

1.3 Multifactor authentication and encryption are becoming the new default

Many still use passwords, but most sites now offer two-factor authentication. Mobile phones and popular operating systems come with biometric identification by default. Identity authentication startup Trusona is making a world without passwords a reality.

Default encryption is on the rise despite nearly all governments protesting it[2]. Today, most popular operating systems, computers, and mobile devices come with built-in, default-enabled disk encryption. More and more websites are using SSL[3] encryption by default although police and government agencies are trying to get rid of default encryption or enabling backdoors in the name of stopping criminals.

2. Computer security landscape

Security is a hard problem. Hard because it is a complex problem with many facets. There are many types of attacks, each targeting a different layer of the ISO/OSI model as shown in Table 1.

Table 1 Some Popular Attack Modes

Layer Name	Popular Protocols	Popular Attack Modes
Layer 7: Application	DNS, DHCP, HTTP, FTP, IMAP, SSH, NTP, SMTP, SNMP, Telnet, TFTP	DNS poisoning, Phishing, SQL injection, spam.
Layer 6: Presentation		
Layer 5: Session	SMB, NFS, Socks	
Layer 4: Transport	TCP (connection-based), UDP (connection-less)	TCP attack, Routing attack, SYN Flooding, Sniffing
Layer 3: Network	IP-v4, IP-v6, ICMP, IPSec	Ping. ICMP Flooding
Layer 2: Data Link	PPTP, Token Ring	ARP spoofing, Mac flooding
Layer 1: Physical		

Ping sweeps and port scans are used for reconnaissance. Sniffing captures packets as they travel through man-in-the-middle attacks intercepting messages intended for a third party. In spoofing, one sets up a fake device and tricks people to send messages

2 Paul Venezia, “The Deep End”, http://www.infoworld.com/article/2946064/encryption/encryption-with-forced-backdoors-is-worse-than-useless-its-dangerous.html, July 13, 2015
3 Roger A. Grimes, “10 security technologies destined for the dustbin”, http://www.infoworld.com/article/2970447/security/10-security-technologies-destined-for-the-dustbin.html

to it. Hijacking means taking control of a session. Advanced persistent threats (APT) and multi-stage attacks are other challenging problems.

There are many components to a network security system that work together to improve the security posture. The most common network security components include firewalls[4], anti-virus software, intrusion detection and prevention systems[5], and virtual private networks[6].

2.1 Attack scenarios

Attackers can take advantage of vulnerabilities in hardware, software, and protocols.

Active attacks are based on an alteration of the original message, or the creation of a false message. An active attack can be an interruption (e.g., masquerading) or modification (e.g., denial of service). In a masquerade, an unauthorized entity pretends to be authorized. (e.g. Due to a lack of authentication, Bob doesn't know if Tom is masquerading as Alice.) Modification results in loss of integrity. A modification attack, in turn, can be a replay or alteration attack. In a replay, if Alice wants to send $100 to Tom's account, Tom captures that message and makes a second transfer of $100. In an alteration, Tom captures Alice's message and alters it to read $200. Denial of service attacks prevent legitimate users from using some services.

In a passive attack, the attacker does not intend to modify but indulges in monitoring the transmission to find out what is happening. There are two types of passive attacks. In Release of Message, the goal is to capture confidential data and put it publicly on the network. In Traffic Analysis, the attacker tries to find similarities between encrypted messages and deduces the original content.

The Speculative Execution attack or Spectre and Meltdown are two newly discovered attack scenarios that exploit critical vulnerabilities in modern processors. These hardware vulnerabilities allow programs to steal data that are currently being processed. While programs are typically not permitted to read data from other programs, these exploits can get hold of secrets stored in the memory of other running programs. This might include passwords stored in a password manager or browser, personal data such as photos, emails, instant messages and even business-critical documents. Unlike usual malware, Meltdown and Spectre are hard to distinguish from regular benign applications. These exploits do not leave any trace in a log file. Meltdown and Spectre work on personal computers, mobile devices, and in the cloud. Depending on the cloud provider's infrastructure, it might be possible to steal data from other customers. Intel has responded to this disclosure in terms of both software patches and firmware updates.

[4] "Managed Firewall Simplify and Streamline the Management and Monitoring of Your Firewall Device", https://www.secureworks.com/capabilities/managed-security/network-security/managed-firewall

[5] "Managed IDS/IPS Two Devices You Shouldn't Be Without", https://www.secureworks.com/capabilities/managed-security/network-security/managed-ids-ips

[6] Chey Cobb, "Ensuring Network Security with a VPN (Virtual Private Network)", http://www.dummies.com/programming/networking/ensuring-network-security-with-a-vpn-virtual-private-network/

Meltdown breaks the most fundamental isolation between user applications and the operating system. If a computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. Luckily, there are software patches against Meltdown. Spectre, on the other hand, breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre.

Speculative Execution is a legitimate procedure that may inadvertently create an opportunity for an attack. In order to improve performance, many CPUs may choose to speculatively execute instructions based on assumptions that are considered likely to be true. During speculative execution, the processor would be verifying these assumptions. If they are valid, then the execution continues. If they are not valid, then the execution is unwound, and the correct execution path can be started based on the actual conditions. It is possible for this speculative execution to have side effects which, if not restored when the CPU state is unwound, can lead to information disclosure.

The "bounds check bypass attack" (Variant 1) allows malicious code to circumvent bounds checking features built into most binaries. Even though the bounds check fails, the CPU will speculatively execute instructions after the bounds checks, and can access memory that the code could not normally access. When the CPU determines the bounds check has failed, it discards any work that was done speculatively; however, some changes to the system can be still observed (in particular, changes to the state of the CPU caches). Malicious code can detect these changes and read the data that was speculatively accessed. [16, 17, 18]

"The branch target injection attack" (Variant 2) uses the ability of one process to influence the speculative execution behavior of code in another security context running on the same physical CPU core. Modern processors predict the destination for indirect jumps and calls that a program may take and start speculatively executing code at the predicted location. The tables used to drive prediction are shared between processes running on a physical CPU core, and it is possible for one process to influence (pollute) the branch prediction tables of another process or kernel code. In this way, an attacker can cause speculative execution of any mapped code in another process, in the hypervisor, or in the kernel, and potentially read data from the other protection domain using techniques like Variant 1.

This vulnerability can be fixed either by a CPU microcode update from the CPU vendor, or by applying a software mitigation technique. This mitigation may be applied to the operating system kernel, system programs and libraries, and individual software programs, as needed. [16,19, 20]

[16] "Hacker News", http://hn.premii.com/#/article/16073874
[17] "CVE-2017-5753", http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5753
[18] "CVE-2017-5715", http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5715
[19] "CVE-2017-5754", https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754
[20] "Meltdown and Spectre", https://meltdownattack.com/

2.2 Intrusion detection

An intrusion detection system (IDS) is the modern-day equivalent to the burglar alarm; it constantly monitors the network to look for suspicious activity and can be configured to notify security administrators of any suspected intrusion. An Intrusion Protection System (IPS) goes a step further by sending an alert and preventing the attempted intrusion, say by dropping traffic if a malicious act is detected. Based on the location in a network, an IDS can be host-based, network-based or application-based [2, 8, 13, 18].

Host-based IDSs, typically software, are installed on host computers and are used to analyze traffic received by the host. For example, an antivirus device may detect unwelcome traffic and log it for further analysis. Host-based systems might also monitor the OS, system calls, audit logs, and error messages on the host system. While network probes can detect an attack, only host-based systems can determine whether the attack was successful. Additionally, host-based systems can record what the attacker performed on the compromised host.

Network-based IDSs (NIDS) use strategically positioned probes to monitor and analyze all traffic on the target network. While host-based detection cannot detect a ping sweep or a port scan across multiple hosts, network-based IDSs can easily detect such reconnaissance attacks. Network-based sensors generate an alert when these reconnaissance attacks are discovered. As network speeds increase, so must the capabilities of the intrusion detection probes. As the network grows, more probes can be added to ensure proper coverage and security. None of these systems were effective in pinpointing attacks quickly; they were generally used as forensic tools to examine security incidents ex post facto.

Depending on how they function, NIDS can be divided into two types: (a) Behavior-based (or, anomaly-based or statistical) IDS, and (b) Signature-based (or, pattern-based) IDS, also known as misuse detection.

In behavior-based systems, the IDSs try to detect intrusions via deviations from normal or expected behavior. Here, IDSs make a profile of every user during normal operation. When a deviation of this normal behavior is detected, the IDS triggers its alarm. This type of IDS can detect the type of intrusion that has no record of its previous occurrence. In that sense, behavior-based systems can detect new type of attack patterns. A large number of false alarms are a problem with this system.

In signature-based systems, the IDSs maintain a database of known exploits and their attack patterns (also called signatures). While analyzing network packets, if the IDS finds any pattern match to one of those known attack patterns, then it triggers an alarm. This type of IDS needs to analyze every packet in the network as it looks for known attack patterns. This type of IDS produces less number of false positives. Since there are many network-based exploits coming on each month, these need to be updated frequently.

While both techniques can be effective in real-time, both suffer from significant limitations: behavior-based techniques break down under heavy traffic or sudden traffic bursts and signature-based techniques cannot guard against unknown

intrusions. State-of-the art anomaly detection methods can identify deviations from normal behavior, but they do not offer guidance on what to do next.

Application-based IDS appears to be the latest novelty in the intrusion detection field. These systems fall into three sub-categories:

- Application self-protection (with OWASP[7] being the most well-known Web application security project)
- Web application firewall (WAF[8])
- Dynamic application security testing (DAST[9])

When it comes to web applications, the majority of companies do not have resources to fix vulnerabilities. A good option is to apply an automated process combining WAF and DAST. Being a passive security system, firewall itself is not capable of recognizing bad content in the traffic. Redirecting the flow to DAST and receiving an "alert" about malicious traces, firewall memorizes the rules and collects them. In its turn, DAST produces reports on application security vulnerabilities.

There are a number of commercial intrusion detection systems in the market: Juniper, McAfee, Cisco, Symantec, etc. These IDSs generally do not provide an ideal performance as advertised. Some of the effective host-based open source IDSs are: OSSEC[10], and Tripwire[11]. Some of the popular open source NIDS are: Snort[12], Real SecureNet, Suricata[13], and Bro.

Snort generates thousands of alerts in a small time. A sample is shown in the Table 2 below. A table like this is usually the starting point for the application of machine learning. Each row of such a table describes an observation or alert. The fields (columns) in each observation may include source IP, destination IP, start and end times, protocol type, number of failed login attempts, number of file operations, number of failed connections from the same host, and so on. These fields are also called features (or attributes). A typical data set may contain millions of alerts (rows) and well over fifty features. An alert is nothing but a capture of an event (or events) that occurred at a given time; it does not mean an intrusion has occurred. Our goal is to look for a meaning in these patterns. Certain collections of alerts may indicate some nefarious activity. Collecting alerts from an IDS is just one way of taking the vital signs of a computer system. There are other ways, such as looking at the system calls generated by the operating system [15].

[7] "Welcome to OWASP", https://www.owasp.org/index.php/Main_Page
[8] Chandan Kumar, "5 Open Source Web Application Firewall for Better Security", https://geekflare.com/open-source-web-application-firewall/, September 4, 2016
[9] Ian Muscat, "DAST vs SAST: A Case for Dynamic Application Security Testing", https://www.acunetix.com/blog/articles/dast-dynamic-application-security-testing/, September 6, 2017
[10] "OSSEC: Open Source HIDS SECurity", http://ossec.github.io/index.html
[11] "Tripwire", https://github.com/Tripwire/tripwire-open-source
[12] "Snort", https://www.snort.org/
[13] "Suricata", https://suricata-ids.org/

Table 2 A Sample Data Set

T	Time	src_port	Dest port	src_ip	dest_ip	Severity
Adobe Products Violation	Sep 01 '14 00:17:50	Medium	Large4	46.178.180.36	33.85.222.155	High
Aggressive Aging	Sep 01 '14 00:23:00	Medium	Large4	15.226.10.38	25.46.150.139	Low
Apache Server Protection Violation	Sep 01 '14 04:17:11	Medium	Large4	53.197.36.240	20.213.101.107	High
Application Servers Protection Violation	Sep 01 '14 04:23:00	Medium	Large4	21.69.151.82	3.138.44.87	High
BACKDOOR: SSH Server Running on Non-Standard Port	Sep 01 '14 08:34:55	Medium	Large5	2.93.222.122	27.131.204.222	Low
BitTorrent: DHT Tracker Communications (UDP)	Sep 01 '14 08:41:36	Medium	Large5	51.105.11.21	15.40.86.223	High
Content Protection Violation	Sep 01 '14 08:47:00	Medium	Large1	19.39.134.253	19.177.132.149	High

3. Modern face of machine learning

Banks and others in financial industry use Machine Learning (ML) to gain insights from data, and to prevent fraud. The insights can also identify investment opportunities, or help investors know when to trade. ML can also identify clients with high-risk profiles.

Machine Learning posits that computers can learn from data without being explicitly programmed to perform specific tasks. In fact, data is key when it comes to building systems using ML. The adage that more data means better models is true when it comes to ML. The input to a ML model could be structured or unstructured data, network data, click-stream data or behavior data and the output is a score or a class label.

Data sets are growing larger. As the volume, velocity, and variability of data streams increase, so does the challenges. Even within a single network, the most basic characteristics – such as bandwidth, duration of connections, and application mix – can exhibit immense variability, rendering them unpredictable. While tools often

work well with thousands of records and a few megabytes of data, they do not scale up well while handling real-world problems, measured in gigabytes or even terabytes of data.

As models are exposed to new data, they incrementally adapt. They learn from previous model stages to produce reliable, repeatable decisions and results. It is a science that is not new – but one that has gained fresh momentum [5, 21, 23, 25].

3.1 Data collection

Table 2 is an example of an unlabeled data set. Traditionally, a security expert examines each row of this table and assigns a label YES for a breach and NO, otherwise. Such labelled data, created laboriously, constitutes the training data for a supervised machine learning algorithm. Then the algorithm learns to classify the rows into two categories: intrusive and benign. Thus, a well-crafted anomaly detection algorithm may be able to generalize (from what it has learned from the training data) and detect an intrusion when presented with data it has not seen earlier [2, 24, 25].

Broadly speaking, there are three stages in the application of ML to solve any problem: preprocessing, training and validation of a model, and post-processing. The training and validation stage has been automated with excellent algorithms and opensource codes (e.g. Scikit-learn[14], Tensor Flow[15]) to implement these algorithms. The bulk of the time and effort is normally spent during the pre- and post-processing (evaluation) stages where the purpose is to gain insight as such human intervention is still required.

A significant challenge in devising an evaluation plan is the lack of publicly available datasets, such as those available in, say, image processing. The two publicly available datasets that – the DARPA/Lincoln Labs packet traces [16, 17], and the KDD Cup dataset derived from them [12] – are almost two decades old, and no longer adequate for any current study.

3.2 Pre-processing

Raw data is often unusable for ML purposes. Data wrangling is the process of cleaning and unifying messy and complex data sets for easy access and analysis. This process typically includes manually converting/mapping data from one raw form into another format to allow for more convenient consumption and organization of the data. There may be missing data, inconsistent data use (e.g., a cardinality feature may contain “North”, “north”, and “N”, all identical in meaning), and numeric data with non-numeric characters, among many other possible problems. This step also involves combining multiple data sources to a single usable source and normalizing data so that all feature values are within a standard range, say in [0,1].

[14] “scikit-learn”, http://scikit-learn.org/stable/
[15] “About TensorFlow”, https://www.tensorflow.org/

For illustration, assume that we have a labelled data set, comprised of 41 columns and millions of rows. To detect an intrusion, we might use a classification algorithm and make a prediction of the class label. A first step of the procedure would be to download the data and examine it. To read the data and the labels we can use Python's pandas.read_csv function and process the resulting data frame using the following process_data function.

```
def process_data(X, y):
    X = X.drop(41, 1)
    X[1], uniques = pandas.factorize(X[1])
    X[2], uniques = pandas.factorize(X[2])

    num_examples = 10**6
    X = X[0:num_examples]
    y = y[0:num_examples]

    X = numpy.array(X)
    y = numpy.array(y).ravel()

    return X, y
```

All this function does is drop the label field (column 41) from X and turn categorical features in columns 1 and 2 into integers, then picks the first million rows and returns the resulting numpy arrays. This idea can be used if one is trying to map source IP's to some numerical values, thus:

```
df['ID'] = pd.factorize(df.SrcIP)[0]
```

Result

```
        SrcIP | ID
192.168.1.112 |  0
192.168.4.118 |  1
192.168.1.112 |  0
192.168.4.118 |  1
192.168.5.122 |  2
```

The next step is feature engineering [3, 19]. Feature engineering is the process of using domain knowledge to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. The table above shows only six features: time, src port, dest port, src ip, dest ip and severity. (What the table did not show is whether or not the alert is a result of a breach of security.) A security expert may feel that the features

selected may not suit her objective. To develop a user profile, subjective features like host ID, time of log-in, commands used, data entry modalities (speed of typing, keyboard Vs mouse usage) may be more relevant. To profile a program, objective features like system calls generated, resources used (CPU time, Memory, Buffers, etc.) may be more appropriate. In the context of intrusion detection, anomalous actions often happen in bursts rather than as isolated events. Due to this reason, time-based features like (a) the number of flows to unique source/destination IP addresses inside the network in the last T seconds to/from the same destination/source and (b) the number of flows from the source IP to the same destination port in the last T seconds, [4] occur.

Another rule of machine learning is to use a training set with instances drawn from all classes in equal proportions. Also, these algorithms perform better when trained with large data sets from each class. One finds only a few anomalies in a large data set. A machine trained with large chunks of normal data (positive examples) performs better in recognizing normal data and it would be foolhardy to expect it to perform well on abnormal data (negative examples).

Dealing with unbalanced datasets entails strategies such as improving classification algorithms or balancing classes in the training data during preprocessing. The later technique is preferred as it has wider application. The main objective of balancing classes is to either increase the frequency of the minority class or decrease the frequency of the majority class using a variety of techniques like under-sampling the majority class, over-sampling the minority class, synthetic minority over-sampling (SMOTE) – where a subset of data is taken from the minority class as an exemplar and new instances are created [9]. This author had used an Earth model to synthetically create a data set of minority class (underground nuclear explosions) while discriminating explosions from earthquakes [16].

3.3 Model building and validation

To make a prediction, one can use any of the classifiers available in the library. For example, logistic regression is used in the next section.

3.4 Evaluation of classification models

Evaluation of the classification model is an important post-processing step. In this connection, there are several important issues that need to be addressed:

(a) What is the purpose of model evaluation? We use a model evaluation procedure to estimate how well a model will generalize to out-of-sample data. We also need a model evaluation metric to quantify model performance.

(b) What are some of the evaluation procedures?

   i. A simple way to evaluate a model is to split the data set into a training set and a test set. This split is best done with packaged s/w tools built

into Scikit or TensorFlow. While fast and simple, this step also gives a better estimate of out-of-sample performance, although it is still a high-variance estimate.

```
# split X and y into training and testing sets
#from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,
random_state=0)
```

ii. K-fold Cross Validation: This systematically creates "K" training splits and averages the results. This runs K-times slower, but gives even a better estimate of out-of-sample performance, although it is still a high-variance estimate.

iii. The model itself can be evaluated depending on the type of problem the model is solving. In regression problems, the usual metrics are mean absolute error, mean square error and the root mean square (RMS) error. If it is a classification problem, accuracy is the traditional metric. The question to be answered is: Can we predict an intrusion status given some measurements on system health?

```
# train a logistic regression model on the training set
#from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

logreg.fit(X_train, y_train)

# make class predictions for the testing set

y_pred_class = logreg.predict(X_test)
```

(c) Now calculate the classification accuracy which is defined as the percentage of correct predictions

```
# calculate accuracy
from sklearn import metrics
print(metrics.accuracy_score(y_test, y_pred_class))
0.692708333333
```

(d) Null accuracy is the accuracy that could be achieved by predicting the most frequent class (here, normal with no intrusion). For brevity, this calculation is not shown here.

(e) For the binary classification (normal Vs intrusive) model being built here from 192 (rows) alerts, only the first 28 results are shown below:

```
# print the first 25 true and predicted responses
from __future__ import print_function
print('True:', y_test.values[0:25])
print('Pred:', y_pred_class[0:25])
```

```
True: [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0]
Pred: [0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

(f) The Confusion Matrix gives a more complete picture of how the classifier is performing. Also, it allows one to compute various classification metrics, and these metrics can provide valuable guidance in model selection. Here, this is a 2x2 matrix because there are 2 response classes. Every observation in the testing set is represented in exactly one box. The basic terminology used is as follows:

- **True Positives (TP):** The model *correctly* predicted that there IS an intrusion
- **True Negatives (TN):** The model *correctly* predicted that there is NO intrusion
- **False Positives (FP):** The model *incorrectly* predicted that there IS intrusion (a “Type I error”)
- **False Negatives (FN):** The model *incorrectly* predicted that there is NO intrusion (a “Type II error”).

```
#IMPORTANT: first argument is true values, second argument is
predicted values
print(metrics.confusion_matrix(y_test, y_pred_class))
```

```
[[118  12]
 [ 47  15]]
```

```
# save confusion matrix and slice into four pieces
confusion = metrics.confusion_matrix(y_test, y_pred_class)
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
```

n=192	Predicted: 0	Predicted: 1	
Actual: 0	TN = 118	FP = 12	130
Actual: 1	FN = 47	TP = 15	62
	165	27	

The confusion matrix can be used to calculate several performance metrics of the classifier, as discussed below:

Classification accuracy, defined as (TP + TN)/(TP + TN + FP + FN), is the easiest metric to understand.

```
print((TP + TN) / float(TP + TN + FP + FN))
print(metrics.accuracy_score(y_test, y_pred_class))
0.692708333333
```

Other metrics can be calculated likewise by replacing the first line of the above code snippet by the appropriate formula, as shown below:

Sensitivity = (TP)/(TP + FN) = 0.241935483871 tells how sensitive the classifier is in detecting positive (intrusive) instances. That is, how often the prediction is correct when the actual value is positive?

Specificity = (TN)/(TN + FP) = 0.907692307692 tells how often the prediction is correct when the actual value is negative?

False Positive Rate = (FP)/(TN + FP) = 0.0923076923077 tells how often is the prediction incorrect when the actual value is negative.

Precision = (TP)/(TP + FP) = 0.555555555556 tells how often is the prediction correct when a positive value is predicted. That is, how "precise" is the classifier while predicting positive instances?

3.5 Which metrics to use?

The choice of metric depends on the objective in modeling. For spam filtering (positive class is "spam"), one optimizes for precision or specificity because false negatives (spam goes to the inbox) are more acceptable than false positives (non-spam is caught by the spam filter). For a fraud detector (positive class is "fraud"), one optimizes for sensitivity because false positives (normal transactions that are flagged as possible fraud) are more acceptable than false negatives (fraudulent transactions that are not detected). In intrusion detection applications, a false positive, requires spending expensive analyst time examining the reported incident only to eventually realize that it reflects benign activity and can quickly render the NIDS ineffective.

False negatives, on the other hand, have the potential to cause serious damage to an organization: even a single compromised system can seriously undermine the integrity of the IT infrastructure. Systems that aim at minimizing FNs also tend to increase FPs. To guarantee that no terrorist passes through an airport security, you may have to risk frisking a grandmother!

3.6 Adjusting classification threshold

A threshold of 0.5 is normally used in binary classification problems to relate class predictions to probabilities. It is straightforward to print, say the first 10, predicted responses using the command `logreg.predict(X_test)[0:10]` and the corresponding predicted probabilities by the command `logreg.predict_proba(X_test)[0:10,:]` and plot a histogram of the predicted probabilities.

3.7 ROC and AUC

Sensitivity and specificity have inverse relationship. Although both are affected by the threshold, it is possible to study the effect of the threshold by plotting the ROC (Receiver Operating Characteristic) curve, which is a plot of the true-positive rate on the y-axis (i.e., sensitivity) and false-positive rate (i.e., 1 – specificity) on the x-axis, for all possible classification thresholds. For an ideal classifier (high sensitivity and high specificity), the ROC curve hugs the upper left corner of the graph [Fawsett, '06]. This curve can be obtained by running the ROC Curve function from the Scikit-Learn's Metrics module with the true values of the testing set stored in y_test, as the first argument and the predicted probabilities stored in y_pred_prob, (NOT y_pred_class) as the second argument.

The AUC is literally the area under the ROC curve and represents the percentage of the total area under the ROC. A higher AUC value is indicative of a better overall classifier as such AUC is often used as a single-number indicator of the performance of the classifier as an alternative to classification accuracy.

4. Summary

The first part of this paper examined the state-of-the-art of computer security in the light of evolving advances in mobile and cloud computing with special attention to intrusion detection. The second part identified spam filtering and fraud detection where ML scored significant successes. Intrusion detection did not score as well in terms of its large-scale adaptation in commercial products probably because it is not a single problem but a syndrome. It seems that ML is good at identifying known patterns of attack (is this pattern there?) and not so good at identifying evolving attack patterns, especially if the attacker can inspect the models being used. As Richard Hamming famously said, "The purpose of computation is insight, not numbers," ML can be profitably used to gain insight into the operation and interpretation of intrusion detection systems because it can do a lot more, a lot faster.

References

1. F. Abagnale: Talks at Google, https://www.youtube.com/watch?v=vsMydMDi3rI, November 27, 2017
2. E. Aroms: NIST Special Publication 800-94 Guide to Intrusion Detection and Prevention Systems (Idps). CreateSpace, Paramount, CA (2012)
3. Arvin L. Bluma and Pat Langley: Selection of relevant features and examples in machine learning. Artificial Intelligence, 97, 245-271, 1997
4. Varun Chandola, Eric Eilertson, Levent Ert¨oz, Gy¨orgy Simon and Vipin Kumar: Data Mining for Cyber Security. Data Warehousing and Data Mining Techniques for Computer Security, http://minds.cs.umn.edu/publications/chapter.pdf, Springer (2006)
5. Vu Dao and V. Rao Vemuri. Computer Network Intrusion Detection: A Comparison of Neural Networks Methods. Differential Equations and Dynamical Systems (Special Issue on Neural Networks), 2002
6. Tom Fawsett: An Introduction to ROC Analysis. Pattern Recognition Letters, 27, 861-874 (2006)
7. R. Fontugne, J. Mazel, K. Fukuda: Hashdoop: a mapreduce framework for network anomaly detection. IEEE conference on computer communications workshops (INFOCOM WKSHPS). 494–499 (2014)
8. P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez: Anomaly-based network intrusion detection: techniques, systems and challenges. Computers & Security, **28** (1-2), 18-28 (2009)
9. Fithria Siti Hanifah, Hari Wijayanto, Anang Kurnia: SMOTE Bagging Algorithm for Imbalanced Data Set in Logistic Regression Analysis. Applied Mathematical Sciences, 9, 2015
10. G.S. Jang, F. U. Dowla, and V. Vemuri: Application of neural networks for seismic phase identification. Proc. IJCNN 91, Singapore, 899-904 (1991)
11. S. Kamaruddin and V. Ravi: Credit card fraud detection using big data analytics: use of PSOAANN based one-class classification. International Conference on Informatics and Analytics, Pondicherry, August 2016
12. KDD cup data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
13. Nathan Keegan, Soo Yeon Ji, Aastha Chaudhary, Claude Concolato, Byunggu Yu and Dong Hyun Jeong: A survey of cloud-based network intrusion detection analysis. Human-centric Computing and Information Sciences, 6:19 (2016)
14. Raghu Krishnapuram and Anirban Mondal: Upcoming research challenges in the financial services industry: a technology perspective. http://www.idrbt.ac.in/assets/publications/Journals/Volume_01/Chapter_04.pdf, IJBT, 1, 66-84 (2017)
15. Yihua Liao and V. Rao Vemuri: Using text categorization techniques for intrusion detection. Proc. Usenix, San Francisco, August 2002
16. R. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. E. Webster, and M. A. Zissman: Results of the 1998 DARPA offline intrusion detection evaluation. Proc. Recent Advances in Intrusion Detection, 1999
17. R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das: The 1999 DARPA off-line intrusion detection evaluation. Computer Networks, 34(4), 579–595, October 2000
18. Chilukuri K. Mohan and Kishan G. Mehrotra: Anomaly detection in banking operations. http://www.idrbt.ac.in/assets/publications/Journals/Volume_01/Chapter_02.pdf, IJBT, 1, 16-48, 2017
19. Nguyen H. T., Petrović S., Franke K.: A comparison of feature-selection methods for intrusion detection. In. Kotenko I., Skormin V. (eds.) Computer Network Security, MMM-ACNS 2010, Lecture Notes in Computer Science, 6258, Springer, Berlin, Heidelberg (2010)
20. Sanjay Rawat, Arun K. Pujari, V. P. Gulati, V. Rao Vemuri: Intrusion detection using text processing techniques with a binary-weighted cosine metric. Journal of Information Assurance and Security, 2005

21. S.A.R. Shah, B. Issac: Performance comparison of intrusion detection systems and application of machine learning to snort system. Future Generation Computer Systems, Elsevier, ISSN 0167-739X (2017)
22. Robin Somer and Vern Paxon: Outside the closed world: On using machine learning for network intrusion detection. IEEE Xplore, July 8, 2010
23. Bayu Adhi Tama and Kyung-Hyune Rhee: A Detailed Analysis of Classifier Ensembles for Intrusion Detection in Wireless Network. J Inf Process Syst, 13(5), 1203~1212, October 2017
24. C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin: Intrusion detection by machine learning: a review. Expert Systems with Applications, 36(10), 11994-12000 (2009)
25. V Rao Vemuri, (Ed.): Enhancing computer security with smart technology, CRC Press, Auerbach Publications, November 21, 2005.