

Network Intrusion Detection

CSIT375/975 AI and Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

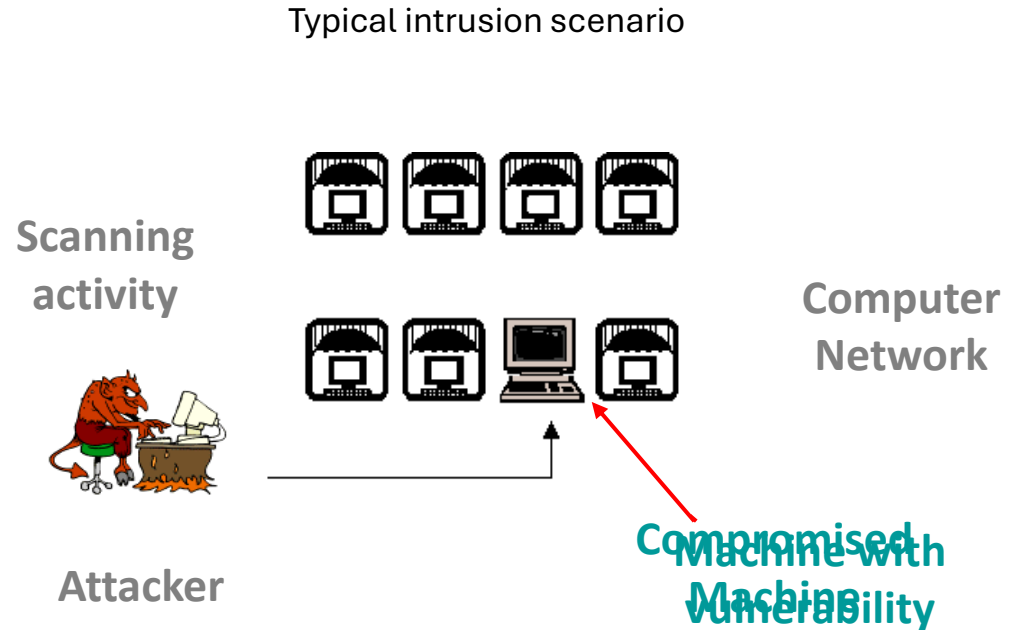
Disclaimer: The presentation materials
come from various sources. For further
information, check the references section

Outline

- Network intrusion detection
- Classify Network Attacks – naïve approach
- Visualizing the analyze network intrusion data

What are network intrusions

- Network Intrusions are illegal actions that attempt to bypass security mechanisms of computer systems. They are caused by
 - Attackers accessing the system from Internet
 - Insider attackers - authorized users attempting to gain and misuse non-authorized privileges



ML for network intrusion detection: motivation

- Traditional network intrusion detection system NIDS tools (e.g. SNORT) are based on signatures of **known attacks**
 - Snort applies rules to monitored traffic and issues alerts when it detects certain kinds of suspicious activity on the network.
- Limitations
 - Signature database has to be manually revised for each new type of discovered intrusion
 - Substantial latency in deployment of newly created signatures across the computer system
 - They cannot detect emerging cyber threats



ML for network intrusion detection: motivation

- Machine Learning and AI-driven techniques can alleviate these limitations – potential to recognize unforeseen attacks.
- Anomaly detection is based on profiles that represent normal behaviour of users, hosts, or networks, and detecting attacks as significant deviations from this profile
 - Major challenges: lack of sufficient amount of high-quality anomalous data
 - Major limitation: possible high false alarm rate, since detected deviations do not necessarily represent actual attacks
 - previously unseen (yet legitimate) system behaviors may also be recognized as anomalies
 - Major approaches: statistical methods, clustering, neural networks, support vector machines, ...

Abnormal Behavior

- Some elements to consider in order to detect anomalous traffic
 - The number of connections to and from a specific host
 - Unusual remote communication ports or unexpected traffic patterns
 - Unusual traffic peaks occurring at particular times of the day (for example, traffic carrying on during the night)
 - Communication bandwidth heavily occupied by particular hosts within the network
- Abnormal behavior in network traffic
 - can be a sign of a security threat, such as a cyber attack or malware infection
 - can also indicate other problems, such as network congestion or faulty equipment

Classify Network Attacks – Naïve approach

- Goal: build a network attack classifier from scratch using basic ML
- Context of the problem
 - In many real scenarios, relevant labelled data might be difficult to come across, and engineering numerical features from the raw data will take up most of the effort
 - Obtaining good training data is a big challenge when using machine learning for security. Classifiers are only as good as the data used to train them, and reliably labelled data is especially important in supervised machine learning
 - Most organizations don't have exposure to a large amount and wide variation of attack traffic, and must figure out how to deal with the class imbalance problem

Classify Network Attacks – Naïve approach

- In the case of classifying network traffic, the task of labelling data often must be carried out by experienced security analysts with investigative and forensic skills, the process is time consuming
- Building high-performing machine learning systems is a process filled with exploration and experimentation
- Next:
 - walk through the process of understanding a dataset and preparing it for processing
 - then present a few algorithms that can be applied to the problem
- Focus: exploration and experimentation process

Exploring the Data

- Dataset: NSL- KDD dataset

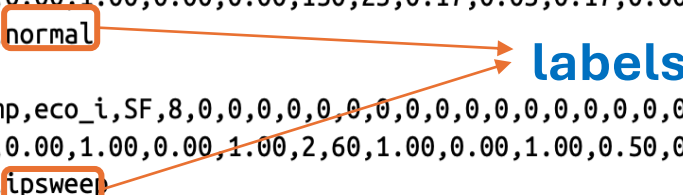
- A benchmark dataset to help researchers compare different intrusion detection methods
- labeled training data: 38 different types of attacks, e.g. ipsweep, guess_passwd...
- Features: first 41 values
 - e.g.: duration, protocol_type, ...

- These attacks belong to **4** general categories:

- **dos**: denial of service
- **r2l**: unauthorized accesses from remote servers
- **u2r**: privilege escalation attempts
- **probe**: brute-force probing attacks

- Task: devise a classifier that categorizes each individual sample as one of five classes: benign, dos, r2l, u2r, or probe.

```
0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,
0.00,0.00,1.00,0.00,0.00,150,25,0.17,0.03,0.17,0.00,0.00,0.00,0.05,
0.00,normal
0,icmp,eco_i,SF,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,21,0.00,0.00,
0.00,0.00,1.00,0.00,1.00,2,60,1.00,0.00,1.00,0.50,0.00,0.00,0.00,
0.00,ipsweep
```



Exploring the Data

- The training dataset contains samples that are labeled with the specific attack, the mapping from attack labels to attack categories is specified in 'training_attack_types.txt'
 - e.g. ftp_write and guess_passwd attacks correspond to the **r2l** category,
 - smurf and udpstorm correspond to the **dos** category, ...
- Preliminary data exploration to find out more about these labels

```
# The directory containing all of the relevant data files
```

```
dataset_root = 'datasets/nsl-kdd'
```

```
category = defaultdict(list)
```

```
category['benign'].append('normal')
```

```
with open(dataset_root + 'training_attack_types.txt', 'r') as f:
```

```
    for line in f.readlines():
```

```
        attack, cat = line.strip().split(' ')
```

```
        category[cat].append(attack)
```

Exploring the Data

Here's what we find upon inspecting the contents of category:

category attack labels

```
'benign': ['normal'],
'probe':  ['nmap', 'ipsweep', 'portsweep', 'satan',
           'mscan', 'saint', 'worm'],
'r2l':    ['ftp_write', 'guess_passwd', 'snmpguess',
           'imap', 'spy', 'warezclient', 'warezmaster',
           'multihop', 'phf', 'imap', 'named', 'sendmail',
           'xlock', 'xsnoop', 'worm'],
'u2r':    ['ps', 'buffer_overflow', 'perl', 'rootkit',
           'loadmodule', 'xterm', 'sqlattack', 'httptunnel'],
'dos':    ['apache2', 'back', 'mailbomb', 'processtable',
           'snmpgetattack', 'teardrop', 'smurf', 'land',
           'neptune', 'pod', 'udpstorm']
```

However, this data structure is not very convenient for us to perform the mappings from attack labels to attack categories, so let's invert this dictionary

Exploring the Data

- What if we want to map from attack labels to attack categories?

```
attack_mapping = dict((v,k) for k in category for v in category[k])
```

Before:

category attack labels

```
'benign': ['normal'],
'probe': ['nmap', 'ipsweep', 'portsweep', 'satan',
          'mscan', 'saint', 'worm'],
'r2l':    ['ftp_write', 'guess_passwd', 'snmpguess',
          'imap', 'spy', 'warezclient', 'warezmaster',
          'multihop', 'phf', 'imap', 'named', 'sendmail',
          'xlock', 'xsnoop', 'worm'],
'u2r':    ['ps', 'buffer_overflow', 'perl', 'rootkit',
          'loadmodule', 'xterm', 'sqlattack', 'httptunnel'],
'dos':    ['apache2', 'back', 'mailbomb', 'processtable',
          'snmpgetattack', 'teardrop', 'smurf', 'land',
          'neptune', 'pod', 'udpstorm']
```

After the mapping:

```
{
    'apache2': 'dos',
    'back': 'dos',
    'guess_passwd': 'r2l',
    'httptunnel': 'u2r',
    'imap': 'r2l',
    ...
}
```

Exploring the Data

- It is always important to consider the class distribution within the data
- In some scenarios, it can be difficult to accurately predict the class distribution of real-life data, but it is useful to have a general idea of what is expected
- For instance, when designing a network attack classifier for deployment on a network that does not contain any database servers, we might expect to see very little SQL attack traffic
- We can consume the data to get the distribution of the `attack_type` and `attack_category`

Exploring the Data

- Read the data and plot the attack_type and attack_category distributions
 - Remind that a Pandas DataFrame is a 2-dimensional data structure, a table with rows and columns
 - read the training data

Read training data

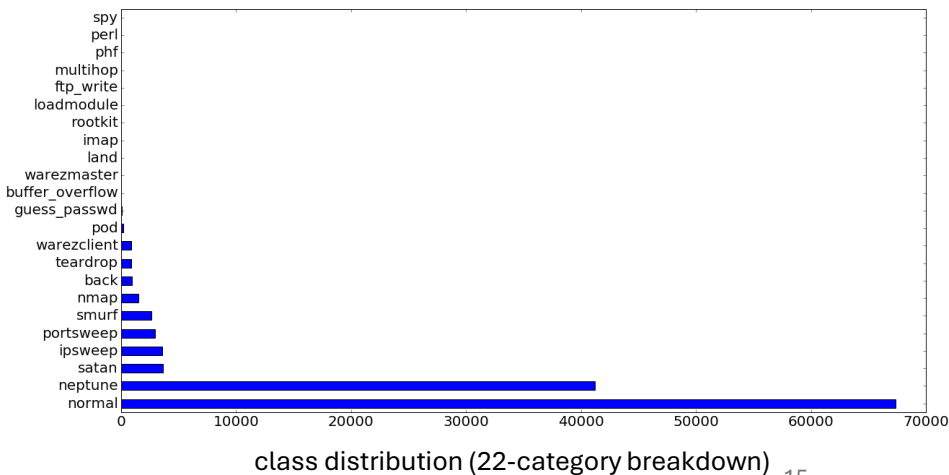
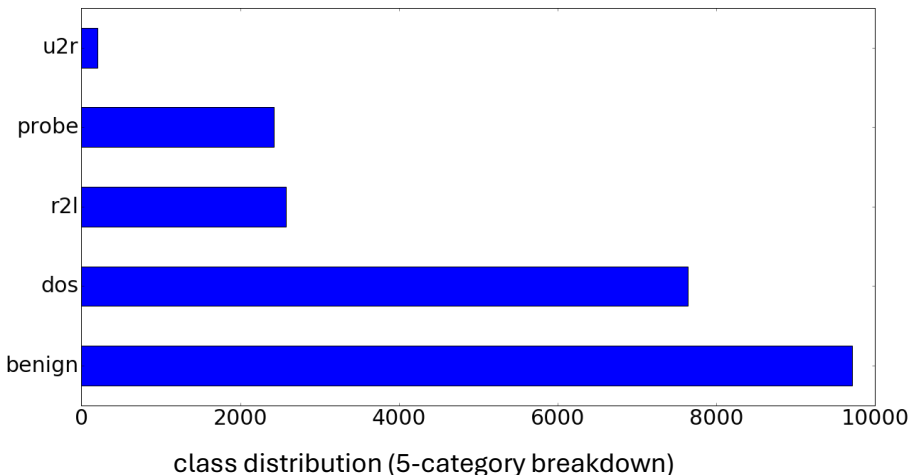
```
train_df = pd.read_csv(train_file, names=header_names)
```

- plot the class distribution

```
train_attack_types = train_df['attack_type'].value_counts()  
train_attack_cats = train_df['attack_category'].value_counts()  
train_attack_types.plot(kind='barh')  
train_attack_cats.plot(kind='barh')
```

Exploring the Data

- Observation: classes are imbalanced
 - if ignore the imbalance, model may learn a lot more about the benign/dos compare to the u2r and r2l
 - undesirable bias in the classifier



Data Preparation

- Split the training and test DataFrames into data and labels

```
train_Y = train_df['attack_category']  
train_x_raw = train_df.drop(['attack_category', 'attack_type'], axis=1)
```

```
test_Y = test_df['attack_category']  
test_x_raw = test_df.drop(['attack_category', 'attack_type'], axis=1)
```

- Encode the categorical (symbolic) variables
 - We will generate a list of categorical variable names and a list of continuous variable names
 - Give a structure containing two keys **continuous** and **symbolic**, each mapping to a list of feature names

```
continuous: [ duration, src_bytes, dst_bytes, wrong_fragment, ... ]  
symbolic:   [ protocol_type, service, flag, land, logged_in, ... ]
```


Data Preparation

- Further split the symbolic variables into nominal (categorical) and binary types
- Then, we use the `pandas.get_dummies()` function to convert the nominal variables into dummy variables
 - Applies **one-hot encoding** to categorical variables (e.g. flag)
 - => create multiple binary variables for each possible value of flag that appears in the dataset
 - For each sample, only one of these variables can have the value 1
 - e.g. if a sample has value flag=S2, its dummy representation (for flag) will be:

```
# flag_S0, flag_S1, flag_S2, flag_S3, flag_SF, flag_SH  
[ 0, 0, 1, 0, 0, 0 ]
```

Data Preparation

- The distribution of the training set features:

	duration	src_bytes	...	hot	num_failed_logins	num_compromised
mean	287.14465	4.556674e+04		0.204409	0.001222	0.279250
std	2604.51531	5.870331e+06		2.149968	0.045239	23.942042
min	0.00000	0.000000e+00		0.000000	0.000000	0.000000
...
max	42908.00000	1.379964e+09		77.000000	5.000000	7479.000000

- We notice something that is potentially worrying
 - The distributions of each feature vary widely
 - e.g. `src_bytes` vs `num_failed_logins`, affect the results as `src_bytes` feature would dominate
- Solution: standardization/normalization

Data Preparation

- Feature scaling: **Standardization** and **Normalization**

- **Standardization**

- Rescales a data series to have a mean of 0 and a standard deviation of 1
- Useful whenever features in the data vary widely in their distribution characteristics
- Standardize feature by removing the mean and scaling to unit variance.

$$X' = \frac{X - \mu}{\sigma}, \text{ where } \mu = \text{mean}(X), \sigma = \text{stddev}(X)$$

- Note that in this case, the values are not restricted to a particular range
- The scikit-learn library includes the `sklearn.preprocessing.StandardScaler` class that provides this functionality

Data Preparation

- For example, let's standardize the duration feature

- Apply standard scaling on it:

```
from sklearn.preprocessing import StandardScaler  
  
standard_scaler = StandardScaler().fit(durations)  
standard_scaled_durations = standard_scaler.transform(durations)  
pd.Series(standard_scaled_durations.flatten()).describe()
```

- We see that the feature has been scaled to have

- a mean of 0
- standard deviation of 1

> # Output:

count	1.259730e+05
mean	2.549477e-17
std	1.000004e+00
min	-1.102492e-01
max	1.636428e+01

Data Preparation

- Normalization

- rescales the data to a small, specified range (min-max scaling)

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} (max - min) + min$$

X_{max} and X_{min} : maximum and minimum value of the feature, resp.

max and min : maximum and minimum value of the **new** feature range, e.g. (0,1) by default

- E.g. If new feature_range = (0, 1)
 - when the value of X is the feature's min_value,
 - numerator = 0 => X' is 0
 - when the value of X is the feature's max_value,
 - numerator = denominator => X' is 1
 - If the value of X is in between => the value of X' is between 0 and 1

Data Preparation

- The `sklearn.preprocessing.MinMaxScaler` class scales a feature from its original range to [min, max].
- This is how `MinMaxScaler` transforms the duration feature between [0,

```
from sklearn.preprocessing import MinMaxScaler
```

```
min_max_scaler = MinMaxScaler().fit(durations)
min_max_scaled_durations = min_max_scaler.transform(durations)
pd.Series(min_max_scaled_duration.flatten()).describe()
```

> # Output:

```
count    125973.000000
mean         0.006692
std         0.060700
min         0.000000
max         1.000000
```

Data Preparation

- Apply **consistent transformations** to both training & test sets
 - i.e., using the same mean, std, etc. to scale the data
- Scikit-learn provides a convenient way to do proper scaling
 - After creating the Scaler and fitting it to the training data
 - We can simply reuse the same Scaler to transform the test data

We finish the data pre-processing phase by standardizing the training and test data:

And...

the data is prepared and ready to go for classifying attacks!

```
# Fit StandardScaler to the training data
standard_scaler = StandardScaler().fit(train_x[continuous_features])

# Standardize training data
train_x[continuous_features] = \
    standard_scaler.transform(train_x[continuous_features])

# Standardize test data with scaler fitted to training data
test_x[continuous_features] = \
    standard_scaler.transform(test_x[continuous_features])
```

Classification

- Choosing a suitable classifier for the task is a **challenging** part
 - Many classifiers could be a good choice, the optimal one might not be obvious
 - Developing machine learning solutions is an iterative process
 - Knowledge and experience with different algorithms can help to develop better intuition to cut down the iteration process
 - but it is rare, even for experienced practitioners, to immediately select the optimal solution for arbitrary machine learning tasks
 - Spending time and effort to iterate on a rough initial solution will bring about useful insights.

Supervised Learning

- What we have in the training data: ~126,000 labeled training samples
- Supervised training methods is a good place to begin, e.g. decision trees
- Confusion matrix resulting from the decision tree classifier
 - Diagonal: # correctly classified samples
 - Size of the test set: total # in the matrix (22,544)
 - row: true class
 - column: predicted class
 - e.g. $a_{04} = 1$: # samples that are of class 0 that were classified as class 4.
 - class index: benign = 0, dos = 1, probe = 2, r2l = 3, and u2r = 4

```
> # Confusion matrix:
[[9357  59  291   3   1]
 [1467 6071   98   0   0]
 [ 696  214 1511   0   0]
 [2325   4   16  219  12]
 [ 176   0   2   7  15]]
```

```
> # Error:
0.238245209368
```

Analysing the result

Similar to the training set, the test data distribution is not balanced across categories

test data distribution

benign	0.430758
dos	0.338715
r2l	0.114265
probe	0.107390
u2r	0.008872

- 43% of the test data belongs to the **benign** class
- 0.8% of the data belongs to the **u2r** class

benign = 0, dos = 1, probe = 2, r2l = 3, and u2r = 4

```
> # Confusion matrix:  
[[ 9357   59  291    3    1]  
 [1467 6071   98    0    0]  
 [ 696  214 1511    0    0]  
 [2325    4   16  219   12]  
 [ 176    0    2    7   15]]
```

- Although only 3.6% of **benign** test samples that were wrongly classified: $(59+291+3+1) / \text{sum}(\text{row}_1)$
- 62% of test samples were classified as **benign**: $\text{sum}(\text{col}_1) / \text{sum}(\text{matrix})$
- Look at **r2l** and **u2r** rows:
 - more samples were classified as **benign** than all other samples in those classes

Why could it be? Could we have trained a classifier that is just more likely to classify samples into the benign class?

Observation

- Going back to see the training data distribution: 0.7% r2l, 0.04% u2r
- Remind the class imbalance in training data, it seems unlikely there would have been enough information for the trained model to learn enough knowledge from the u2r and r2l classes to correctly identify them
- To see whether problems might be caused by the choice of the decision tree classifier, we will try other classifiers, e.g. KNN and SVM
 - KNN (K-Nearest Neighbors), SVM (Support Vector Machine) : supervised learning classifier
 - We use them as a black-box models.

Supervised learning

- Results for KNN

```
> # Confusion matrix:  
[[9455  57  193   2   4]  
 [1675 5894   67   0   0]  
 [ 668  156 1597   0   0]  
 [2346   2   37  151  40]  
 [ 177   0   4   6  13]]
```

```
> # Error:  
0.240951029099
```

- Results for SVM

```
> # Confusion matrix:  
[[9006  294  405   3   3]  
 [1966 5660   10   0   0]  
 [ 714  122 1497   88   0]  
 [2464   2   1  109   0]  
 [ 175   1   0   8  16]]
```

```
> # Error:  
0.278167139815
```

Resulting confusion matrices and error rates show very similar characteristics to the decision tree classifier results

- At this point, it should be clear that continuing to experiment with classification algorithms might not be productive
- What we can try to do is to remedy the class imbalance problem in the training data in hopes that resulting models will not be overly skewed to the dominant classes.

3D Visualization of Network Intrusion Detection Data

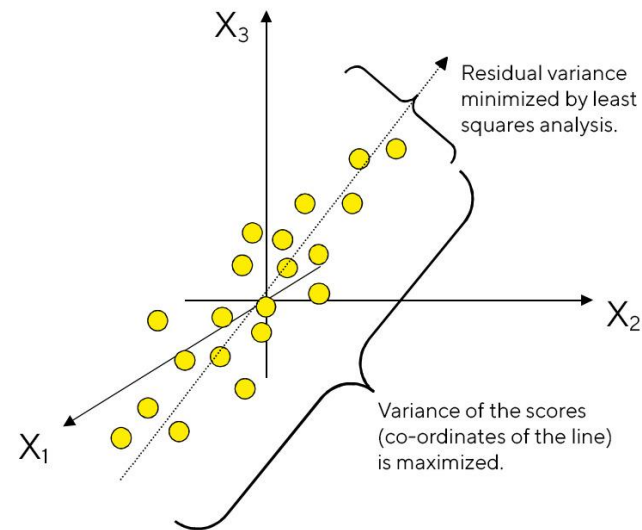
- Motivation to analyze network intrusion dataset
 - Misclassification is a common problem in machine learning
 - A general lack of insight into why such misclassification occurs impedes the improvement of models.
 - The improvement of ML models usually relies on a trial-and-error process, due to the complex nature of ML mechanisms.
 - Many studies have presented improved results when comparing with other techniques
 - There is not much emphasis on understanding the reasons for the improved results or lower misclassification rates.
 - They tend to present numeric results in the form of tables or graphs to compare performances between different ML techniques.
 - The underlying reasons for poor performance on certain attack categories are not usually explained and cannot be perceived clearly or intuitively.
 - An intuitive and explainable analysis of datasets can facilitate the understanding of detection results.
 - ML heavily depends on the characteristics of the training and testing datasets.
 - Explain detection results from the point of view of data.

3D Visualization of Network Intrusion Detection Data

- An interactive 3D visualization
 - The aim of the approach is to provide a visual representation of data from NIDS datasets
 - Portrays the **geometric** relationship between diverse network traffic data records
 - Create a way of examining the likely causes of ML misclassification from a visual perspective.
- The visualization system allows users to interactively navigate through the NIDS data
 - In real-time
 - Visually examine ML classifier decision spaces
 - Observe boundaries where misclassification occurs.

3D Visualization of Network Intrusion Detection Data

- Principal Component Analysis (PCA)
 - PCA finds lines, planes and hyper-planes in the K-dimensional space that approximate the data as well as possible in the least squares sense.
 - A line or plane that is the least squares approximation of a set of data points makes the variance of the coordinates on the line or plane as large as possible.
 - PCA can reduce high dimensionality of NIDS data.
 - To 2D or 3D for visualization.



PCA creates a visualization of data that minimizes residual variance in the least squares sense and maximizes the variance of the projection coordinates.

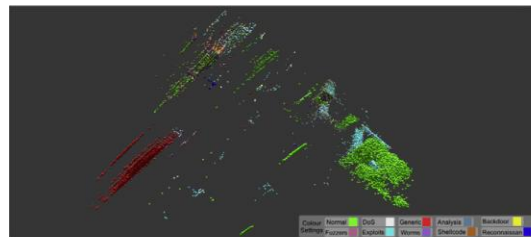
3D Visualization of Network Intrusion Detection Data

- Visually explore NIDS datasets (SVM was the classifier):
 - KDD_No_Boundaries_All_Data.mp4
 - KDD_Binary_Overall.mp4
 - KDD_Multi_Overall.mp4
 - UNSW_No_Boundaries_All_Data.mp4
 - UNSW_Binary_overall.mp4
 - UNSW_Multi_overall.mp4

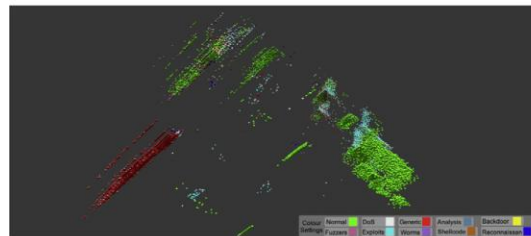
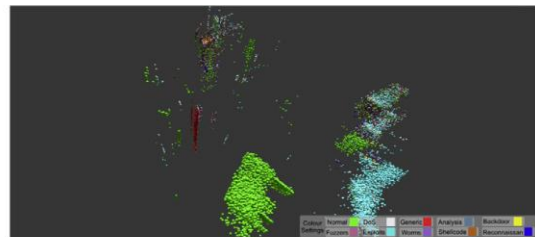
3D Visualization of Network Intrusion Detection Data

- Visually explore **UNSW-NB15**

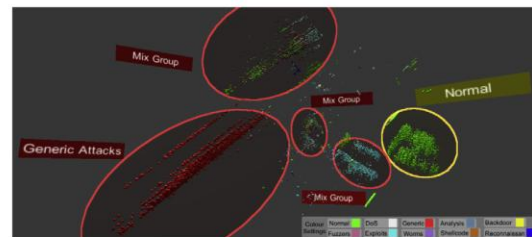
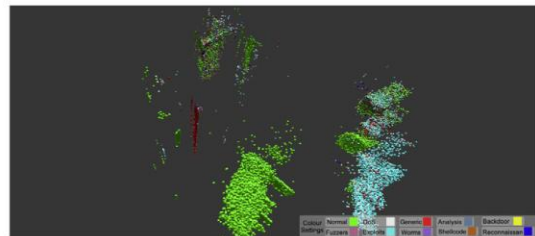
- (a)** shows a visual depiction of a portion of the training set from two different camera viewpoints, where the different network traffic categories have been color coded to visually distinguish them from one another.
 - It can be seen visually that traffic from the same category are typically clustered together.
- (b)** shows the visual representation of part of the testing set from two different camera viewpoints
 - Clearly, characteristics of both training and testing sets are similar.
 - Training a ML model using the training set it is highly likely that the model will be able to detect attacks in the testing set.



(a)



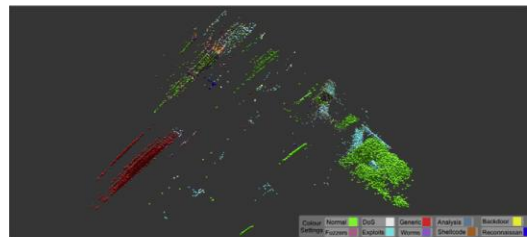
(b)



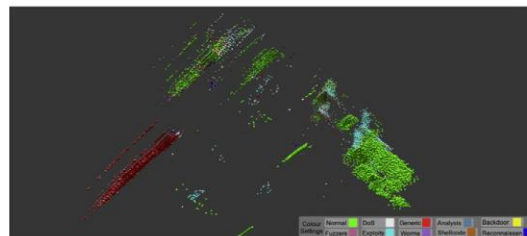
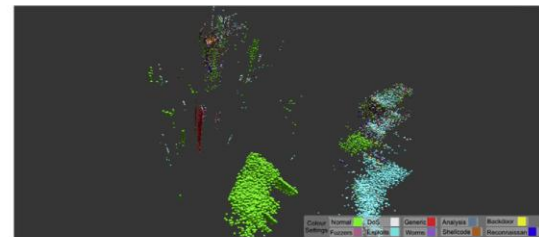
(c)

3D Visualization of Network Intrusion Detection Data

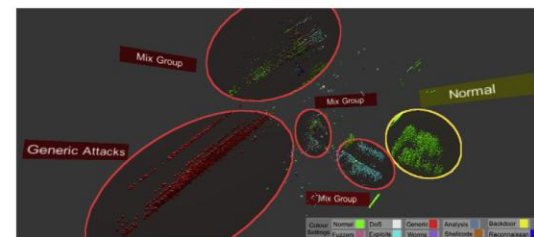
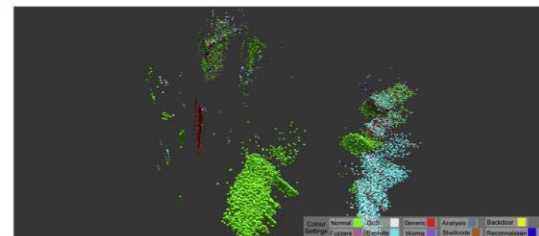
- Visually explore **UNSW-NB15** (continue)
 - **(c)** provides an empirical grouping of data from the training set.
 - The clusters circled in yellow depict sections that mainly only contain normal network traffic, whereas other clusters are circled in red.
 - The traffic for generic attacks are well clustered together.
 - However, the traffic in the mixed clusters is not so well defined as they contain both attacks and normal network traffic.



(a)



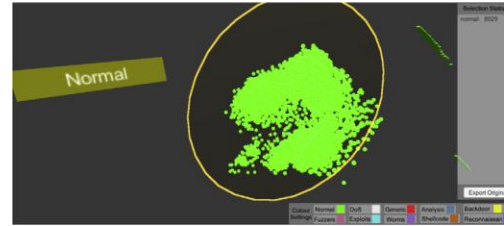
(b)



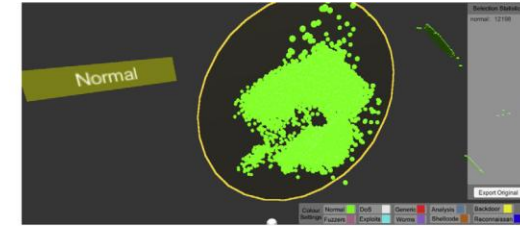
(c)

3D Visualization of Network Intrusion Detection Data

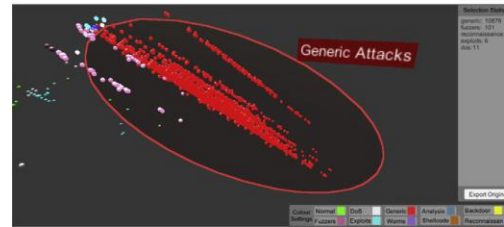
- A closer investigation of **UNSW-NB15**
 - Examples of normal traffic clusters from the training and testing data are shown in (a) and (b), respectively.
 - It can be seen from the figures that the characteristics of the training and testing data are very similar.
 - The implication of this for ML is that traffic in clusters that contain nearly homogeneous records are easier to correctly identify and typically result in high detection performance.
 - (c) and (d) show examples of a cluster from the training and testing data, respectively,
 - Contain mostly generic attacks.
 - The detection performance of most ML models on these generic attacks will be high.
 - The results from several studies appear to support this conclusion.
 - A two-stage approach for generic attacks:
 - recall 96.6%, precision 99.9%



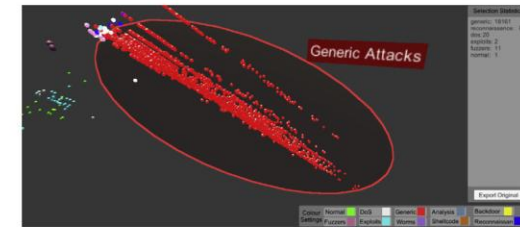
(a)



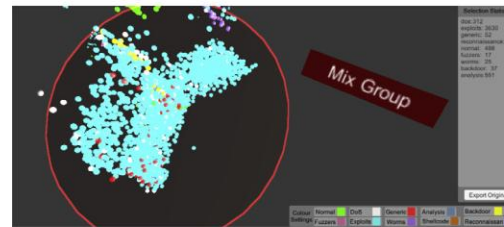
(b)



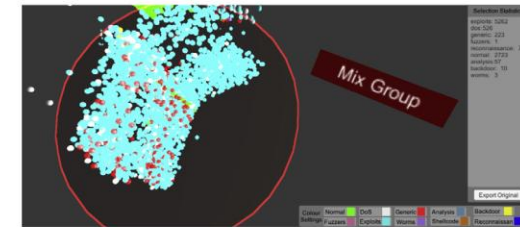
(c)



(d)



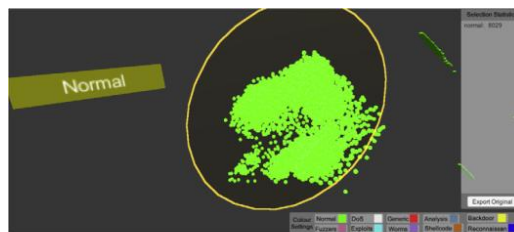
(e)



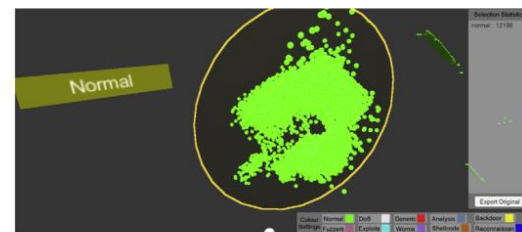
(f)

3D Visualization of Network Intrusion Detection Data

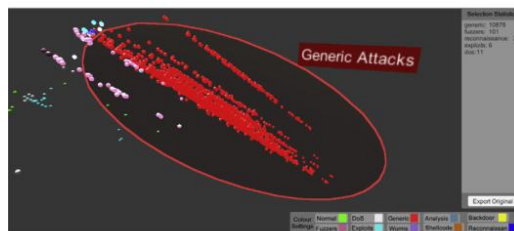
- A closer investigation of **UNSW-NB15**
 - In other sections of the dataset, the traffic is mixed together and there is no clear visual distinction between the different traffic categories within these clusters.
 - (e) and (f) for the training and testing data, respectively.
 - Such sections of the dataset are where ML techniques tend to face difficulties when it comes to the task of correctly identifying the individual categories of the traffic.
 - There is a low number of worm attacks, which is disproportional when comparing the numbers of different traffic in the training set and the testing set.
 - Furthermore, as there is a significant number of exploit attacks within the cluster, it is challenging for ML models to avoid misclassifying other traffic in the cluster as exploits.



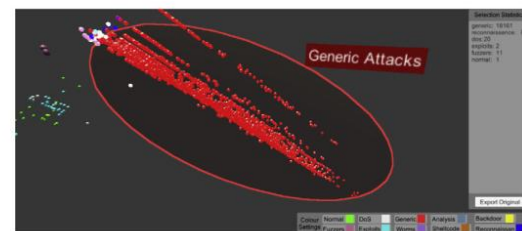
(a)



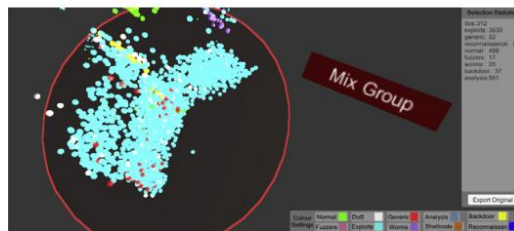
(b)



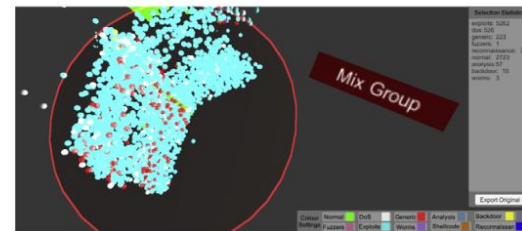
(c)



(d)

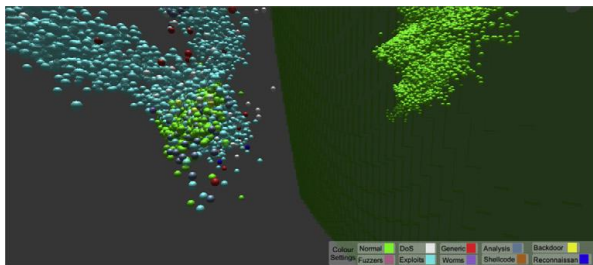


(e)

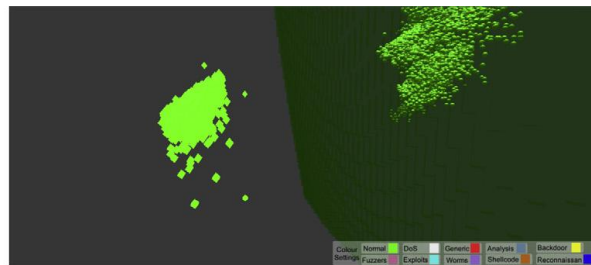


(f)

3D Visualization of Network Intrusion Detection Data



(a)



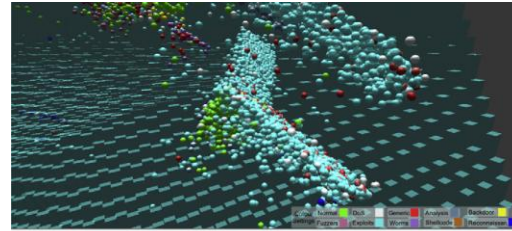
(b)

- A closer investigation of **UNSW-NB15**
 - A close-up of the visualization that clearly shows where this misclassification occurs.
 - In (a), the normal traffic decision space can be seen on the right of the figure and all normal traffic instances within that space should have been correctly classified.
 - However, to the left of the figure there is a cluster of network traffic instances that are located outside the normal traffic decision space, which has a mixture of both normal and abnormal traffic.
 - In (b), all abnormal traffic instances (from (a)) have been removed and the figure highlights the instances of normal traffic that the ML model misclassified as abnormal traffic.
 - It can clearly seen that a large number of normal traffic instances have been misclassified as abnormal traffic.

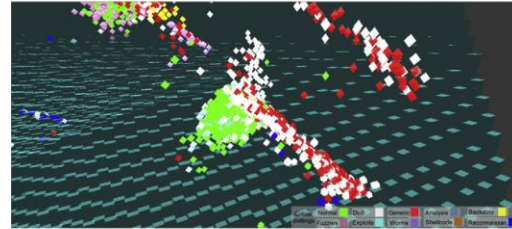
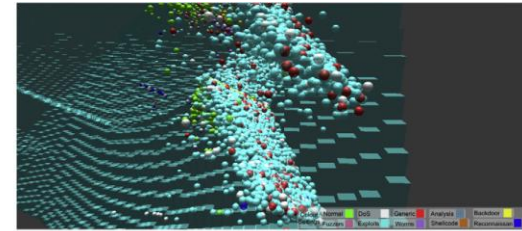
3D Visualization of Network Intrusion Detection Data

- A closer investigation of **UNSW-NB15**

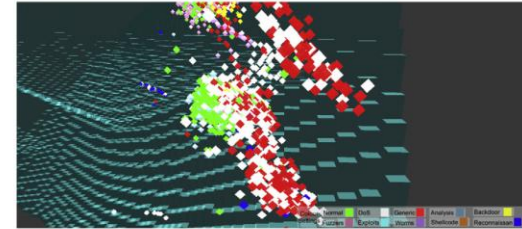
- In some clusters, in which there was a greater proportion of **exploit** attacks than other categories, other traffic would potentially be misclassified as exploits.
- This was verified in the results, as can be seen in the example shown in the figures.
 - Figure (a) are close-ups of part of the exploits decision space from two different camera viewpoints; they show a mixture of exploits and other traffic that lie within the decision space for exploits.
 - In Figure (b), the traffic instances that were misclassified as exploits are highlighted from two different camera viewpoints.



(a)



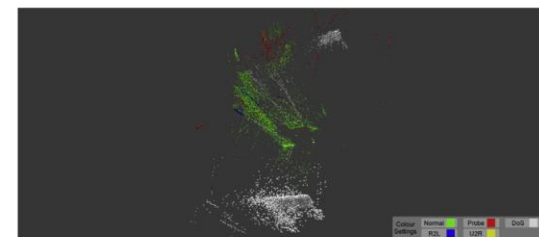
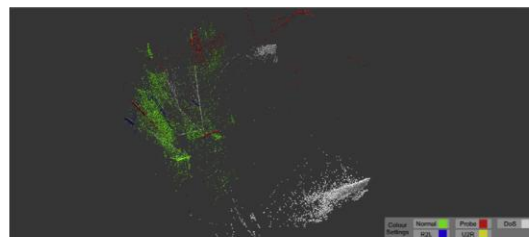
(b)



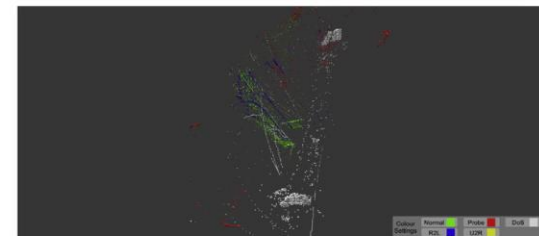
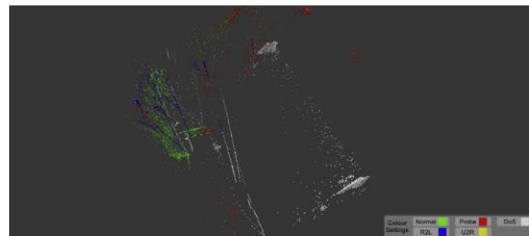
3D Visualization of Network Intrusion Detection Data

- Visually explore **NSL_KDD**

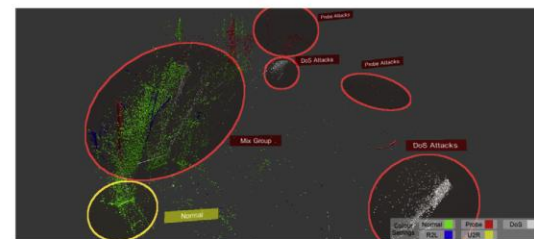
- Figures show the overall 3D visual representation of the NSL_KDD dataset.
 - Figure (a) depicts a portion of the data from the training set,
 - Figure (b) displays part of the data from the testing set, from two different camera viewpoints respectively.
- While the overall visual distributions share relatively similar characteristics, the training and testing sets in the NSL_KDD dataset have more differences between them as compared with the UNSW-NB15 dataset
 - i.e. the characteristics of traffic between the training and testing sets in the UNSW-NB15 dataset exhibited higher similarities.
- Figure (c) shows the empirical grouping of data from the training set.
 - Like the UNSW-NB15 dataset, it also shows that the data can be grouped into several clusters.



(a)



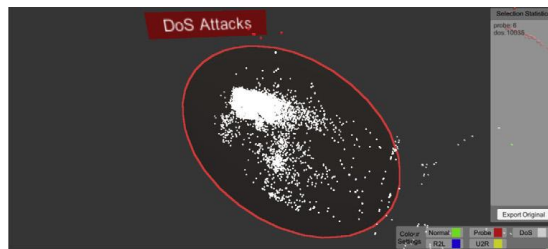
(b)



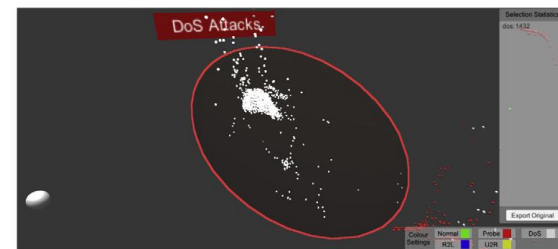
(c)

3D Visualization of Network Intrusion Detection Data

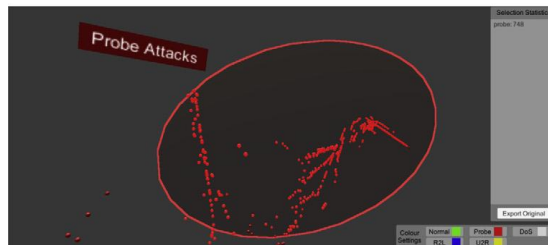
- A closer investigation of **NSL_KDD**
 - From the visual representation, one can observe that the data consists of clusters that contain mainly homogeneous records as well as clusters that contain diverse traffic.
 - Figures (a) and (b) show examples of DoS attack traffic from the training and testing data, respectively, from two different camera viewpoints.
 - It is clear that the display of traffic within the cluster is isolated away from the rest of the traffic.
 - It contains mostly homogeneous DoS attack records.
 - Due to the isolation and clustering of such homogeneous data, ML techniques are expected to perform well when identifying such traffic.



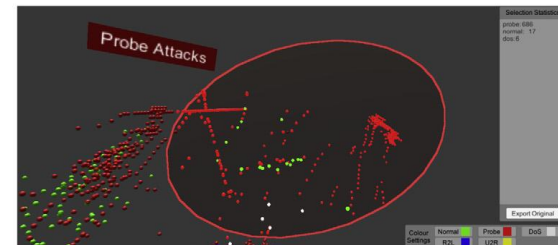
(a)



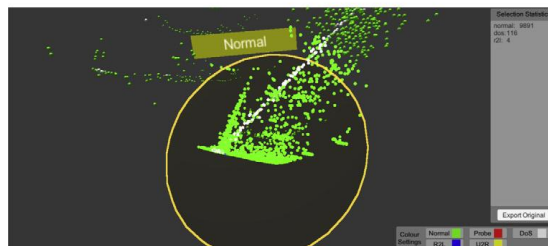
(b)



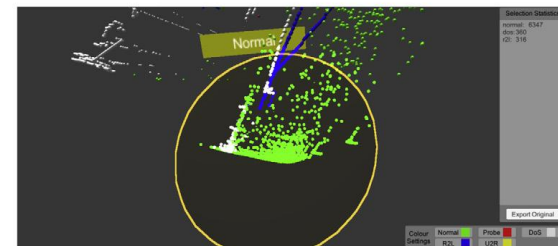
(c)



(d)



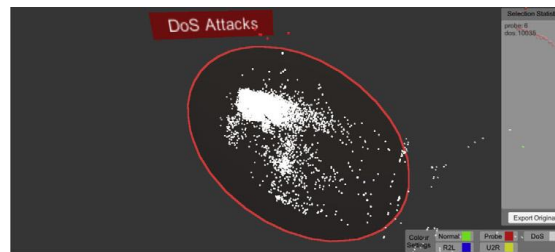
(e)



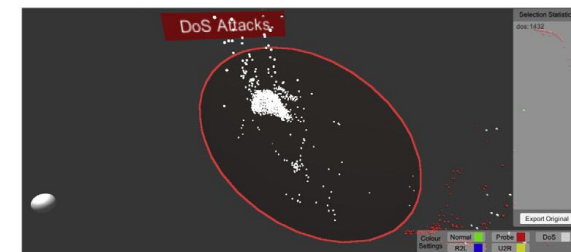
(f)

3D Visualization of Network Intrusion Detection Data

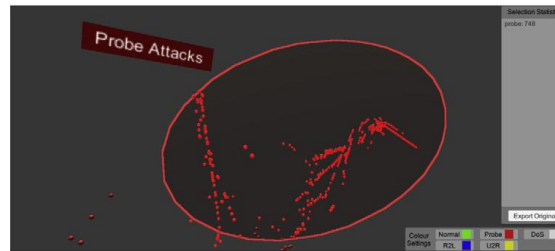
- A closer investigation of **NSL_KDD**
 - Figures (c) and (d) show another example of a cluster containing distinguishable homogeneous traffic
 - containing probe attacks, as from the training and testing data, respectively.
 - Although in (d) there are other traffic near the cluster, such as normal traffic, the majority of traffic inside the cluster is still probe attacks.
 - However, ML models may misclassify normal traffic and other attacks near this cluster as probe attacks when applied to the testing set.



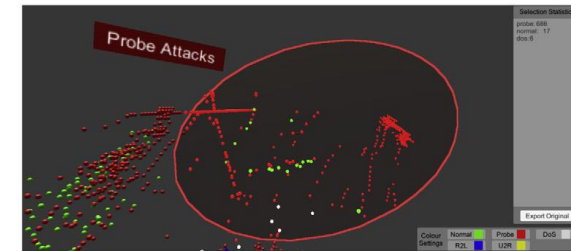
(a)



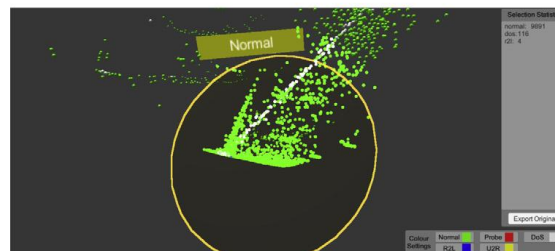
(b)



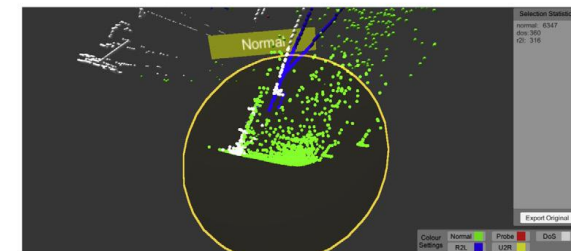
(c)



(d)



(e)

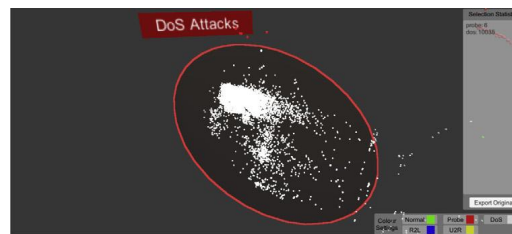


(f)

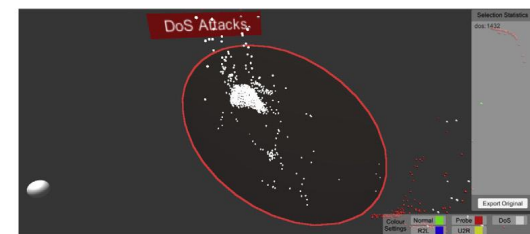
3D Visualization of Network Intrusion Detection Data

- A closer investigation of **NSL_KDD**

- Cyber security experts have identified that the main difficulty faced by ML techniques in the NSL_KDD comes from previously unknown attacks in the testing set [1].
 - Even though the categories of attacks are the same in the training and testing sets, the characteristics of these traffic significantly differ.
- A visual representation of this comparing the training and testing data is shown in (e) and (f), respectively.
 - In the training set in (e), there are almost no R2L attacks within this cluster.
 - On the other hand, there are many R2L attacks in the corresponding cluster from the testing set, as shown in (f).
- ML models should be designed based on the training set.
 - ML techniques that do not consider such abnormal characteristics may perform unsatisfactorily when it comes to detecting such attacks.



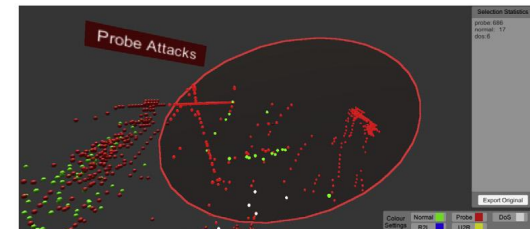
(a)



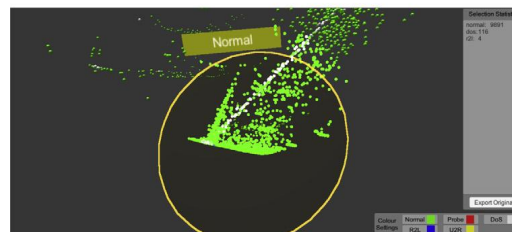
(b)



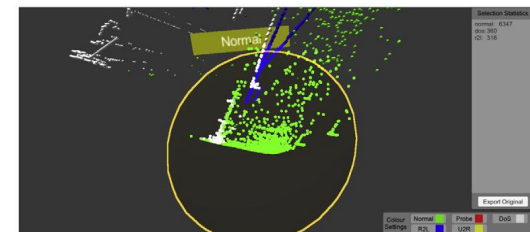
(c)



(d)

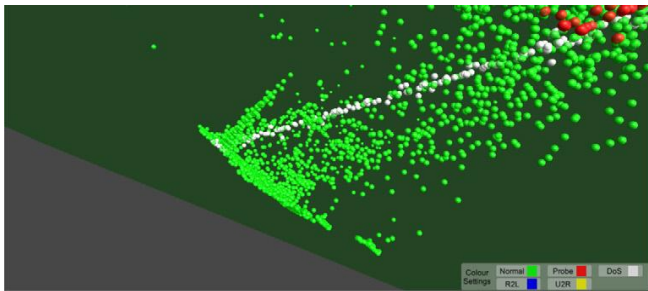


(e)

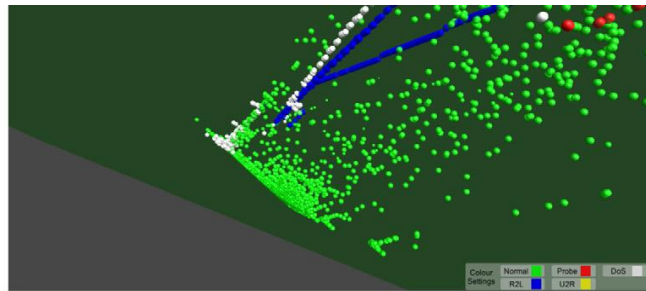


(f)

3D Visualization of Network Intrusion Detection Data



(a)



(b)

- A closer investigation of **NSL_KDD**
 - Visualizing decision boundaries of an SVM model.
 - The SVM model is trained using the training data,
 - If the training data does not adequately represent the network traffic, this can affect the detection accuracy of the ML model.
 - (a) shows part of the normal traffic decision space from the training data,
 - (b) shows part of the normal traffic decision space from the testing data.
 - In figure (b), **R2L** attacks are within the normal traffic decision boundaries in the testing set and they have been misclassified as normal traffic.
 - The reason is because these attacks were previously unknown in the training data.
 - In addition, there are a number of DoS attacks in both (a) and (b).
 - However, while samples are present in the training set, they are misclassified
 - Because they only occupy a small portion of that region in the training set.

3D Visualization of Network Intrusion Detection Data

- Summary of visualization results
 - The visualization approach works well on **UNSW-NB15** and **NSL_KDD** datasets
 - A significant portion of variances in the data are captured by first few principal components,
 - Otherwise, the visualization may result in a situation where no obvious clusters can be observed as there may not be enough variance to differentiate between network traffic instances.
 - The experiments show different geometric characteristics in the visual representation of the **UNSW-NB15** and **NSL_KDD** datasets.
 - The main challenge presented to ML techniques in the **UNSW-NB15** dataset is due to sections that contain clusters with highly heterogeneous data
 - The main challenge in the **NSL_KDD** dataset is due to the abnormal characteristics of unknown attacks.
 - This demonstrates the usefulness of 3D visualization.
 - This allows cybersecurity experts to examine and recognize patterns of incoming network traffic.
 - And also identify the major challenges in a specific network environment.

References

- G. Kim, S. Lee, S. Kim, *A novel hybrid intrusion detection method integrating anomaly detection with misuse detection*, *Expert Syst. Appl.* 41 (4) (2014) 1690–1700, <http://dx.doi.org/10.1016/j.eswa.2013.08.066>.
- *Network Traffic Analysis, Machine Learning and Security Protecting Systems with Data and Algorithms*, Clarence Chio, David Freeman
- Zong, W., Chow, Y.W. and Susilo, W., 2020. *Interactive three-dimensional visualization of network intrusion detection data for machine learning*. *Future Generation Computer Systems*, 102, pp.292-306.