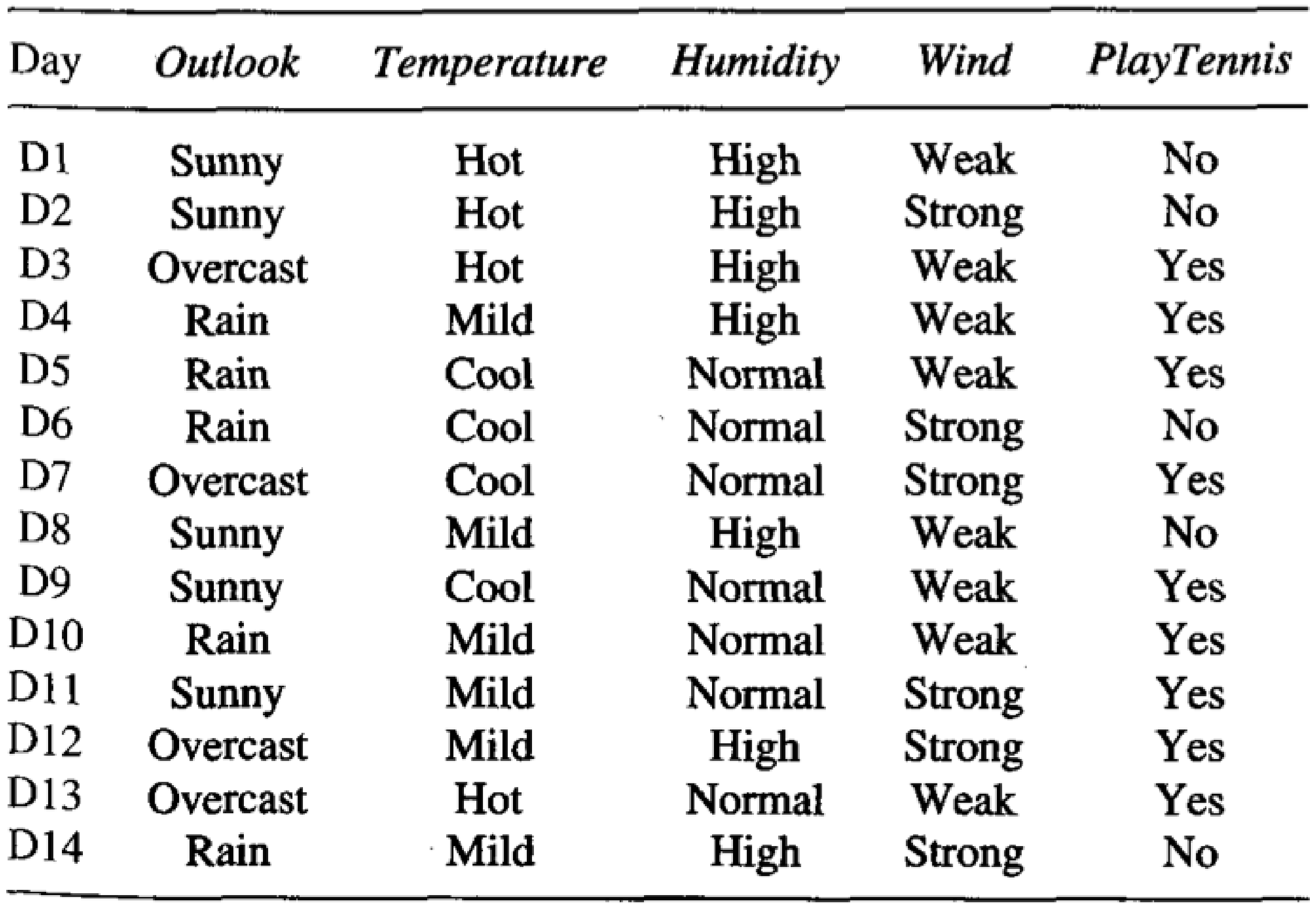
Network Intrusion Detection Case Study

CSIT375 AI for Cybersecurity

Dr Wei Zong

SCIT University of Wollongong

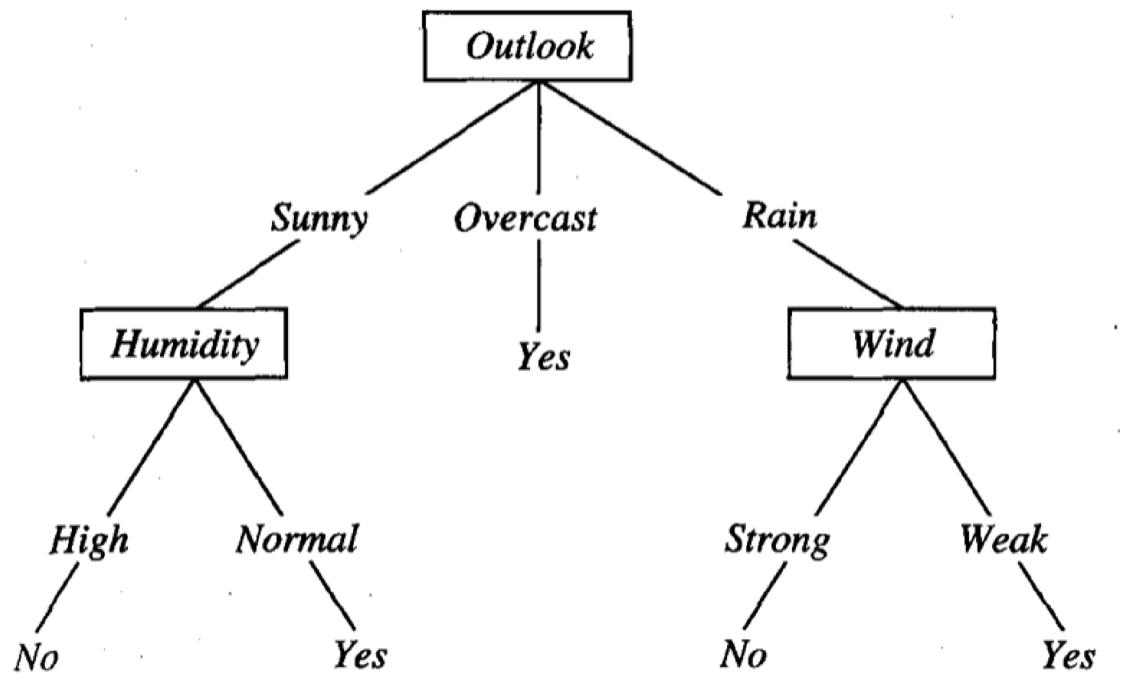
Recap: Decision Tree



Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree Example

- A possible decision tree for the data:



- Each internal node:
  - tests one attribute X_i (such as outlook, humidity, wind…)
- Each branch from a node:
  - selects one value for X_i
- Each leaf node:
  - prediction of Y (yes or no)

What is a good decision tree

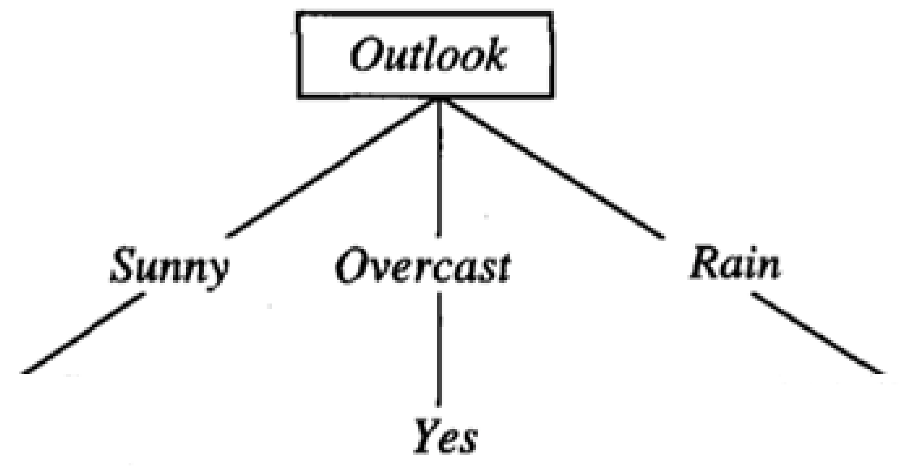
- When we learn a decision tree, we want to determine which attribute in a given dataset of training examples is the most useful for discriminating between the classes to be learned
- We use information gain (IG) which tells us how important a given attribute is to decide the order of attributes for splitting in a decision tree.
  - IG: expected reduction in entropy of target variable Y after split over variable X
- Algorithm for building Decision Tree: ID3
- ID3 employs a top-down manner to construct a decision tree.
- ID3 builds the decision tree from the root node, and at each step, it makes the decision of node to be split based on the information gain.

ID3 Algorithm

- 1. Base entropy is computed based on the distribution of label Y.
- 2. Each attribute is evaluated to get the information gain which determines how well it classifies the training data.
- 3. The best attribute is selected and used as the root node of the tree.
- 4. A descendant of the root node is created for each possible value of this attribute.
- 5. The previous processes are repeated on the remaining attributes to find the best attribute based on the information gain as the descendant node.
- 6. Finally, we form a decision tree.
- (if there is a tie on IG, randomly pick one of the attributes)

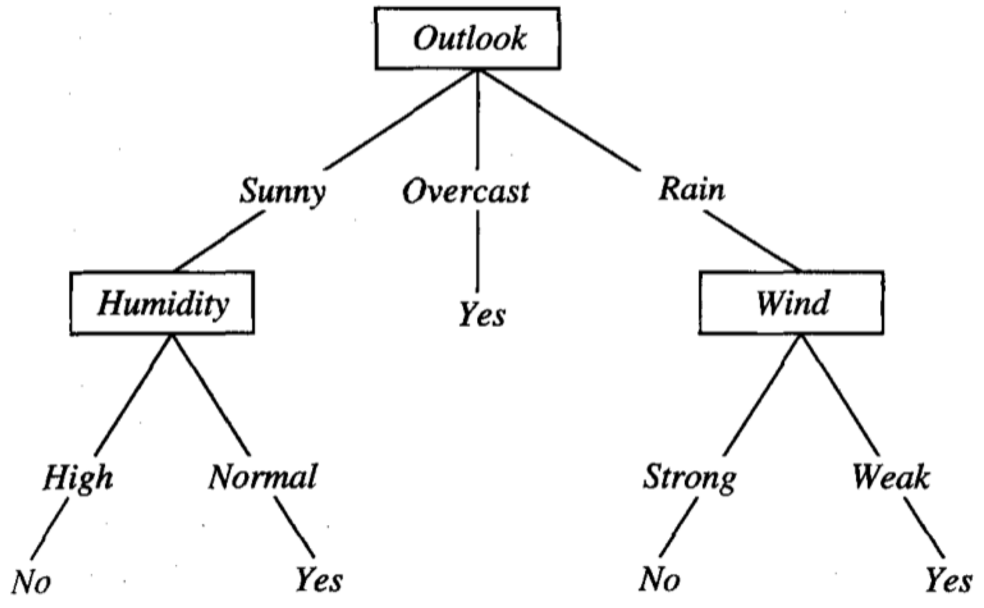
ID3 example

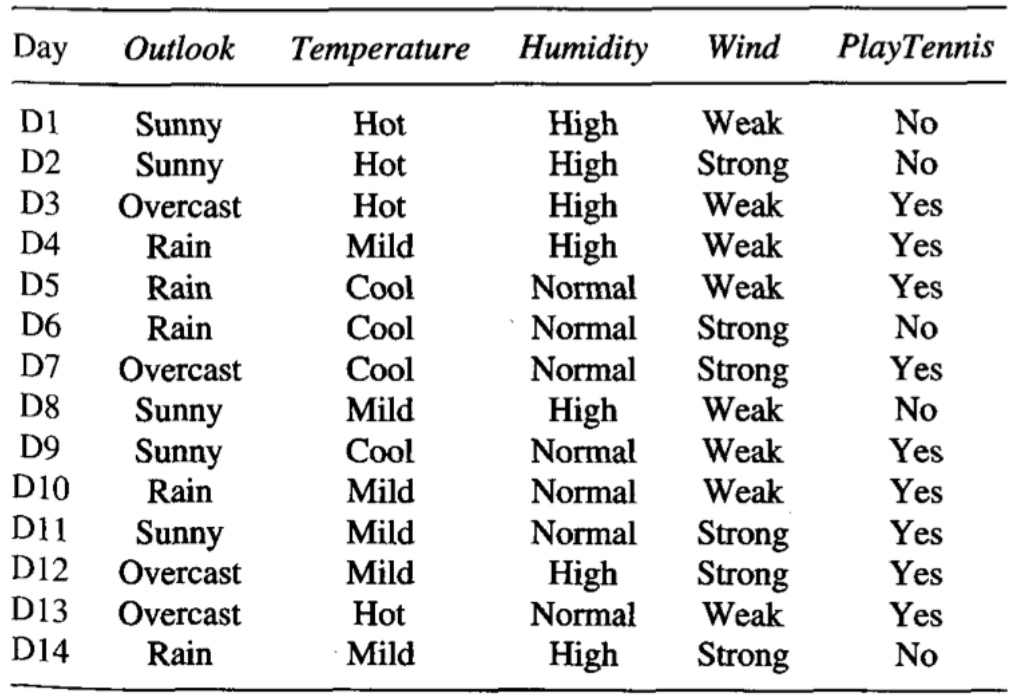
- Among all attributes, outlook has the most information gain.
- We make outlook the root node of the decision tree and its values as the branches.
- Because when outlook=overcast, the labels are all positive, we have a pure leaf node in this tree.



ID3 example

- For each branch of the root node, we repeat the computation of the information gain, and find the best node as the child node.





Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Outline

- Classify Network Attacks –basic approaches
- Two-stage classification
- Dimensionality reduction of network intrusion data
- Visualizing the analyze network intrusion data
    - Identify patterns that potentially cause misclassification

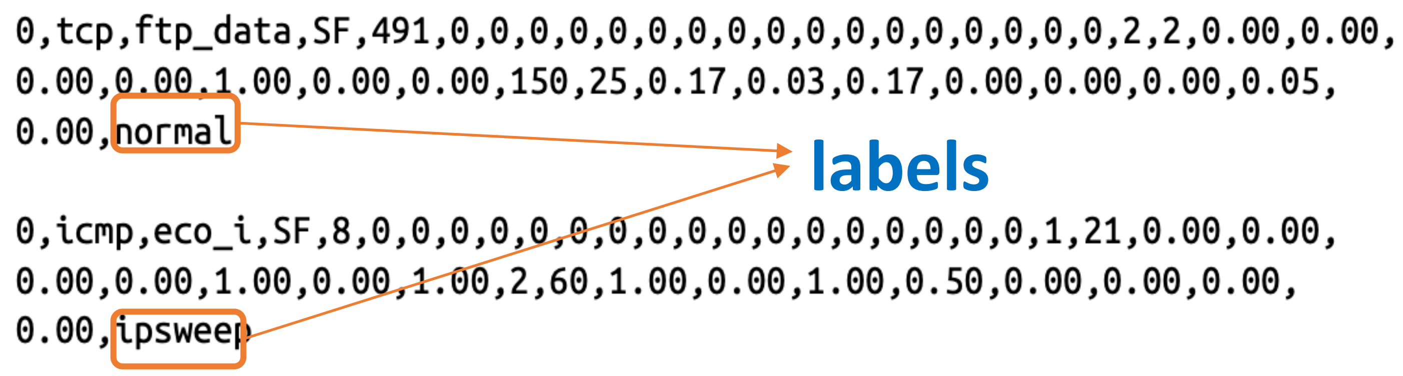
Classify Network Attacks – Case Study

- 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 – Case Study

- 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



```
0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,
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
```

- 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.

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

```python
# 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 sqlattack 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

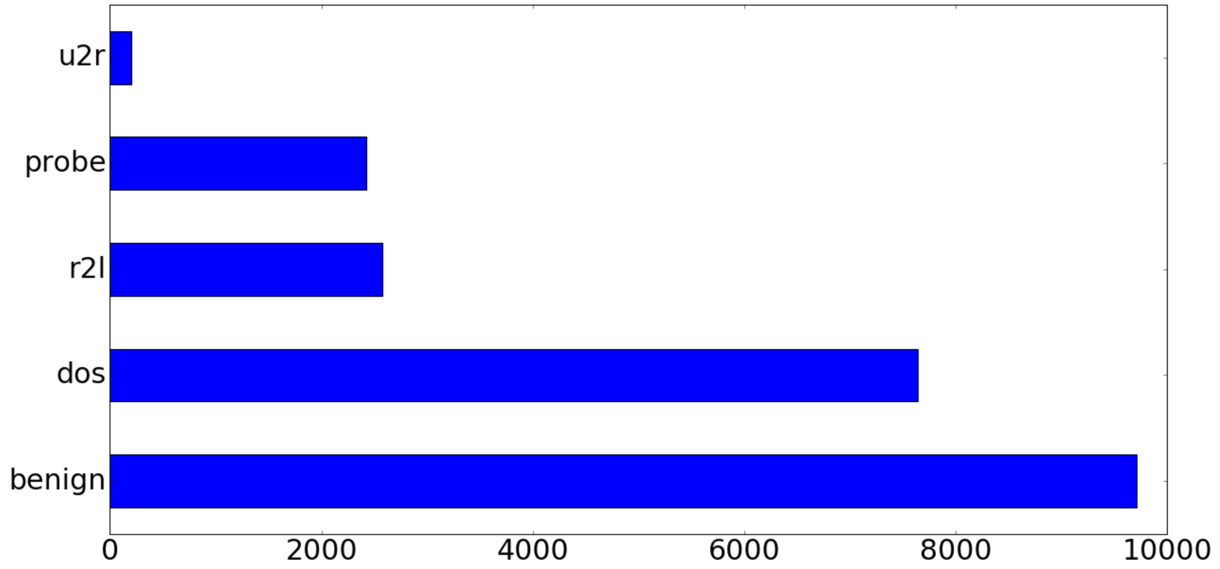
```
# Read training data
train_df = pd.read_csv(train_file, names=header_names)
```

  - plot the class distribution

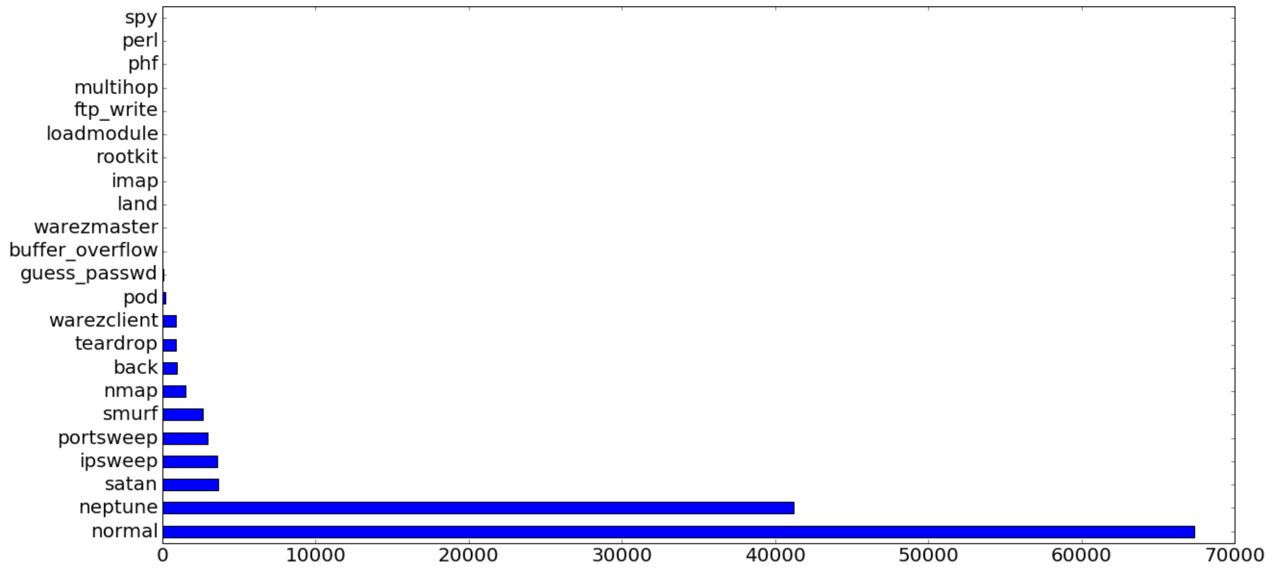
```
train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()
train_attack_types.plot(kind='barh')
train_attack_cats.plot(kind='barh')
```

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



class distribution (5-category breakdown)



class distribution (22-category breakdown)

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, ... ]
```

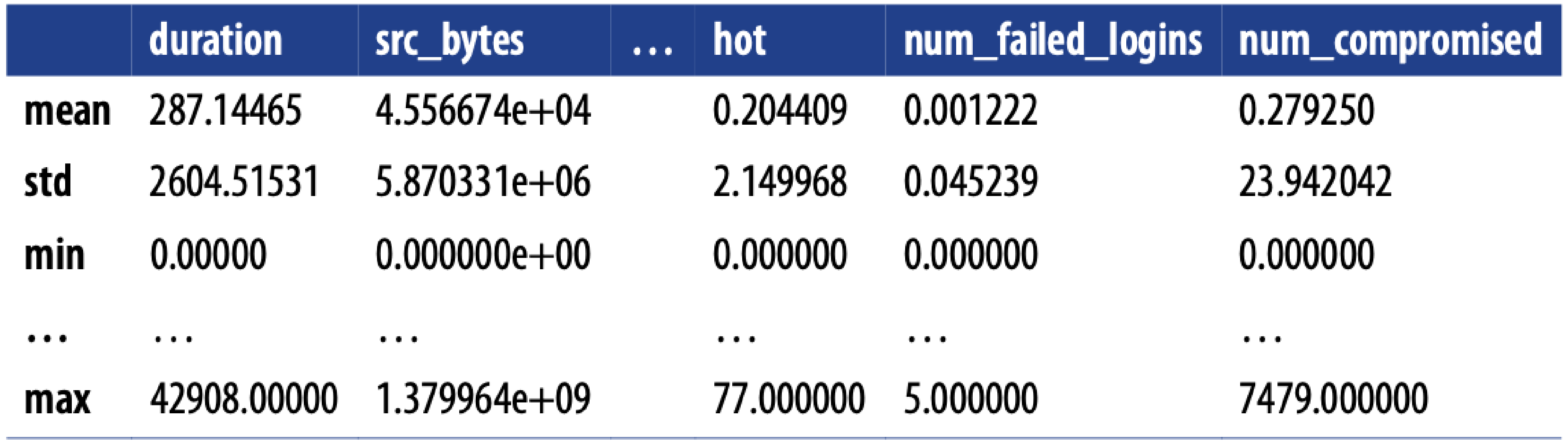
Data Preparation

- 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

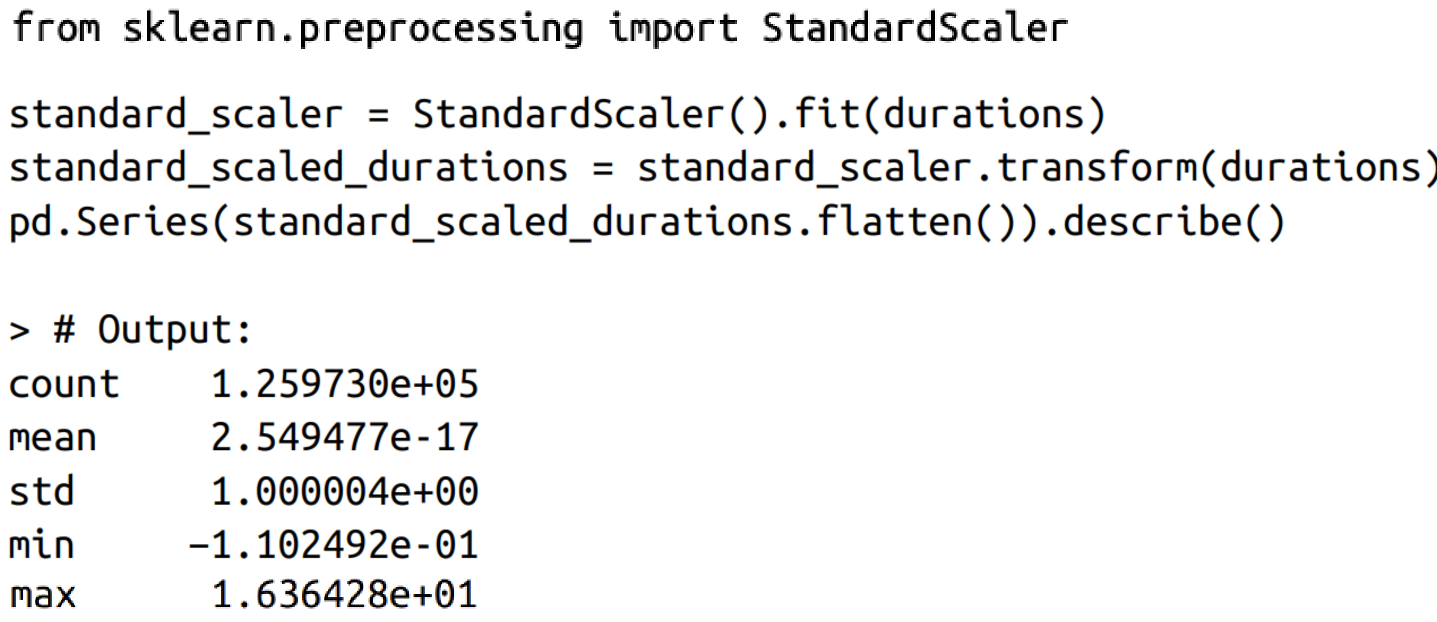
- Feature scaling: Standardization and Normalization
- Standardization
  - rescales a data series to have a mean of 0 and a standard deviation of 1
  - useful whenever features in the data vary widely in their distribution characteristics
  - Standardize feature by removing the mean and scaling to unit variance.

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

  - 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:
- We see that the feature has been scaled to have
  - a mean of 0
  - standard deviation of 1



```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler().fit(durations)
standard_scaled_durations = standard_scaler.transform(durations
pd.Series(standard_scaled_durations.flatten()).describe()

> # 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, 1]

```
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])
```

Multiclass Classification

- Remind that we have a five-class classification problem
  - each sample belongs to one of the following classes: benign, u2r, r2l, dos, probe
- Essentially, a multiclass classification problem can be split into multiple binary classification problems
  - Binary logistic regression
  - Basic SVM.
- Some techniques for achieving multiclass classification
  - one-versus-all: train one classifier per class
  - one-versus-one: train one classifier per class pair

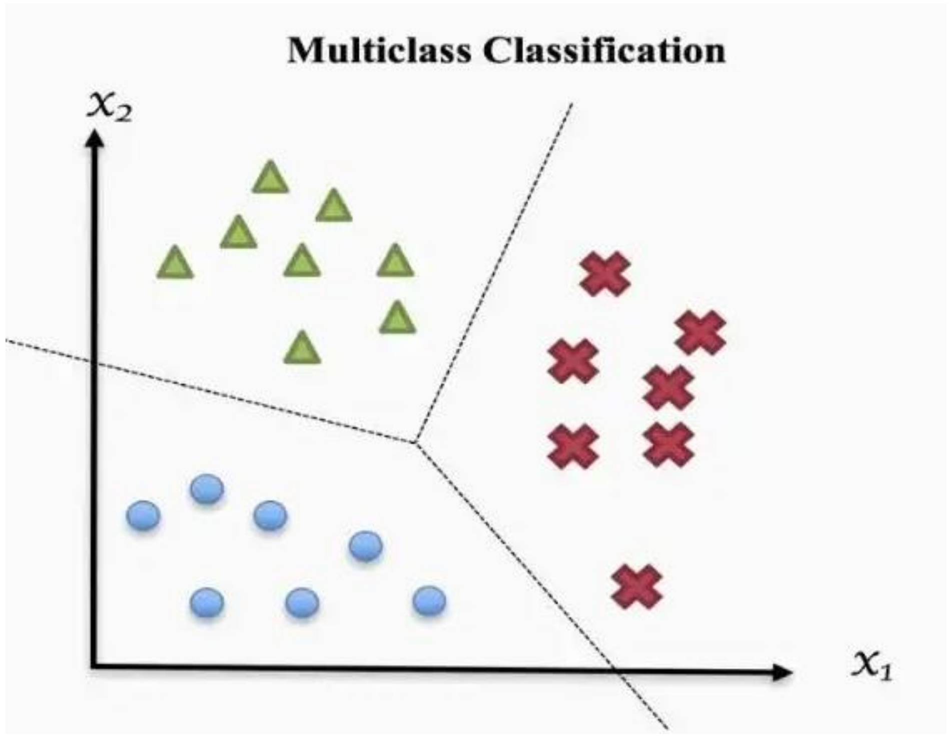
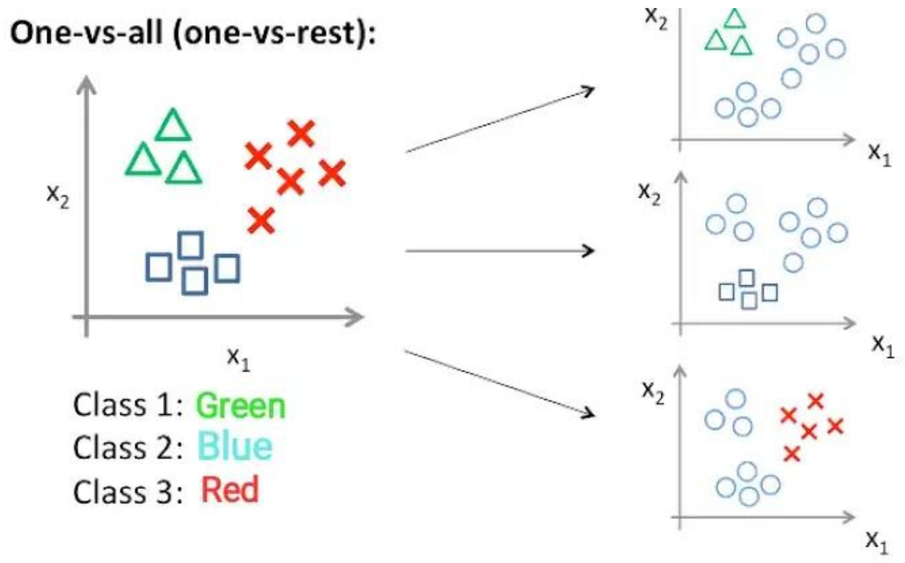


image credit: towardsdatascience

Multiclass Classification

- One-vs-all (one-vs-rest)
  - For the N-class instances dataset, a one-vs-all solution consists of N separate binary classifiers—one binary classifier for each class
  - During training, the model runs through a sequence of binary classifiers, training each to answer a separate classification question
  - At prediction time, the class with highest confidence is selected



Classifier 1:- [**Green**] vs [Red, Blue]
Classifier 2:- [**Blue**] vs [Green, Red]
Classifier 3:- [**Red**] vs [Blue, Green]

Multiclass Classification

- one-vs-one
  - train one binary classifier per class pair
  - for the N-class instances dataset, generate N* (N-1)/2 binary classifier models
- Example: red, blue, black
- **3** binary classifier:
  - Classifier 1: red vs. black
  - Classifier 2: blue vs. black
  - Classifier 3: red vs. blue
- At prediction time, the class which received the most votes is selected

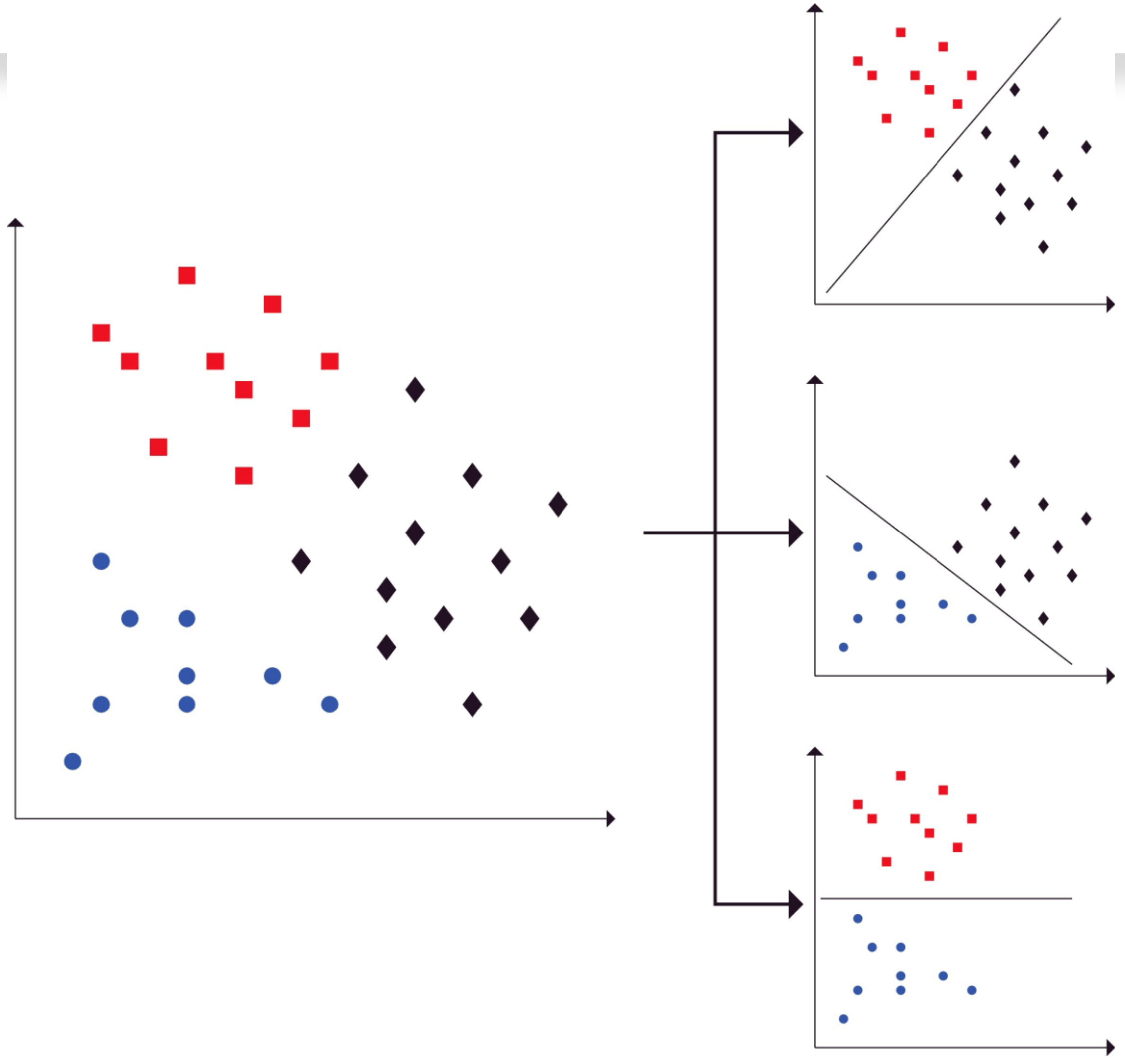


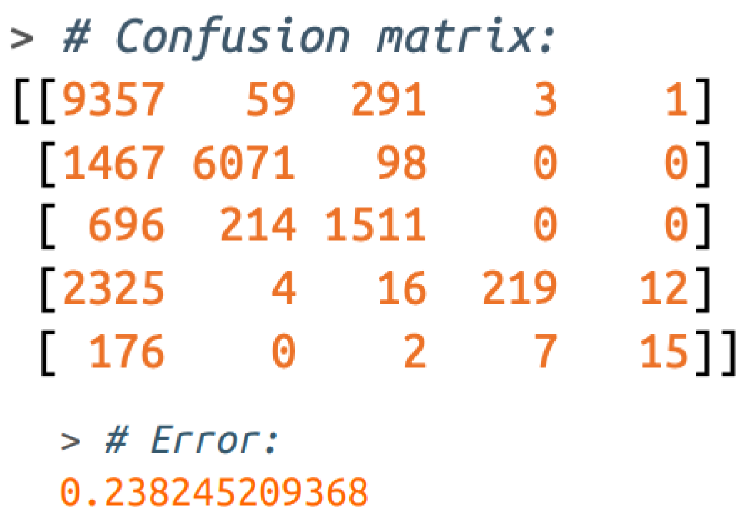
Image Credit: Empowering one-vs-one decomposition with ensemble learning for multi-class imbalanced data

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

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]]
```

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

- Although only 3.6% of benign test samples that were wrongly classified: (59+291+3+1)/ sum(row_1)
- 62% of test samples were classified as benign: sum(col_1) / sum(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): supervised learning classifier
  - Note: we will not dive into the details of KNN; instead, we will use it as a black box for comparison purpose

Supervised learning

- Results for k-nearest neighbors classifier

```
> # 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 linear support vector classifier

```
> # 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

Class Imbalance

- Extremely imbalanced datasets for NIDS cause problems for machine learning.
  - A dataset in which the classification categories are not approximately equally represented is considered to be imbalanced.
  - Certain attacks occur more often than others.
  - Machine learning intrusion detection approaches may perform well at the task of detecting frequent attacks, but are much less effective when it comes to the detection of infrequent attacks, due to the lack of sufficient training data for infrequent attacks.

Class Imbalance

- In general, there are two types of remedies
  - Over-sampling: generating synthetic data points for minority classes
    - a naive way: add random samples to the dataset
    - This would likely pollute the dataset and incorrectly influence the distribution
    - one possible solution: SMOTE (Synthetic Minority Oversampling Technique)
      - algorithms that attempt to generate synthetic data in a way that does not contaminate the original characteristics of the minority class
  - Under-sampling: sampling the overrepresented classes to reduce the number of samples
    - may cause information loss in certain datasets
    - should prioritize the removal of samples that are very similar to other samples

- *Imbalanced-learn* offers a wide range of data resampling techniques for different problems

A two-stage approach

- A two-stage approach for imbalanced intrusion detection datasets
  - The underlying notion behind this approach is to alter the dataset into majority and minority malicious activity classes, and to apply classification algorithm on them separately to produce different models for detection.
  - The purpose of this is to improve the overall detection rate of minority classes and to reduce the error rate.
  - Different classifiers can be used for each of the two-stages.
    - Decision tree, logistic regression, deep neural network, etc.
    - Random forest is used in experiments.

Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

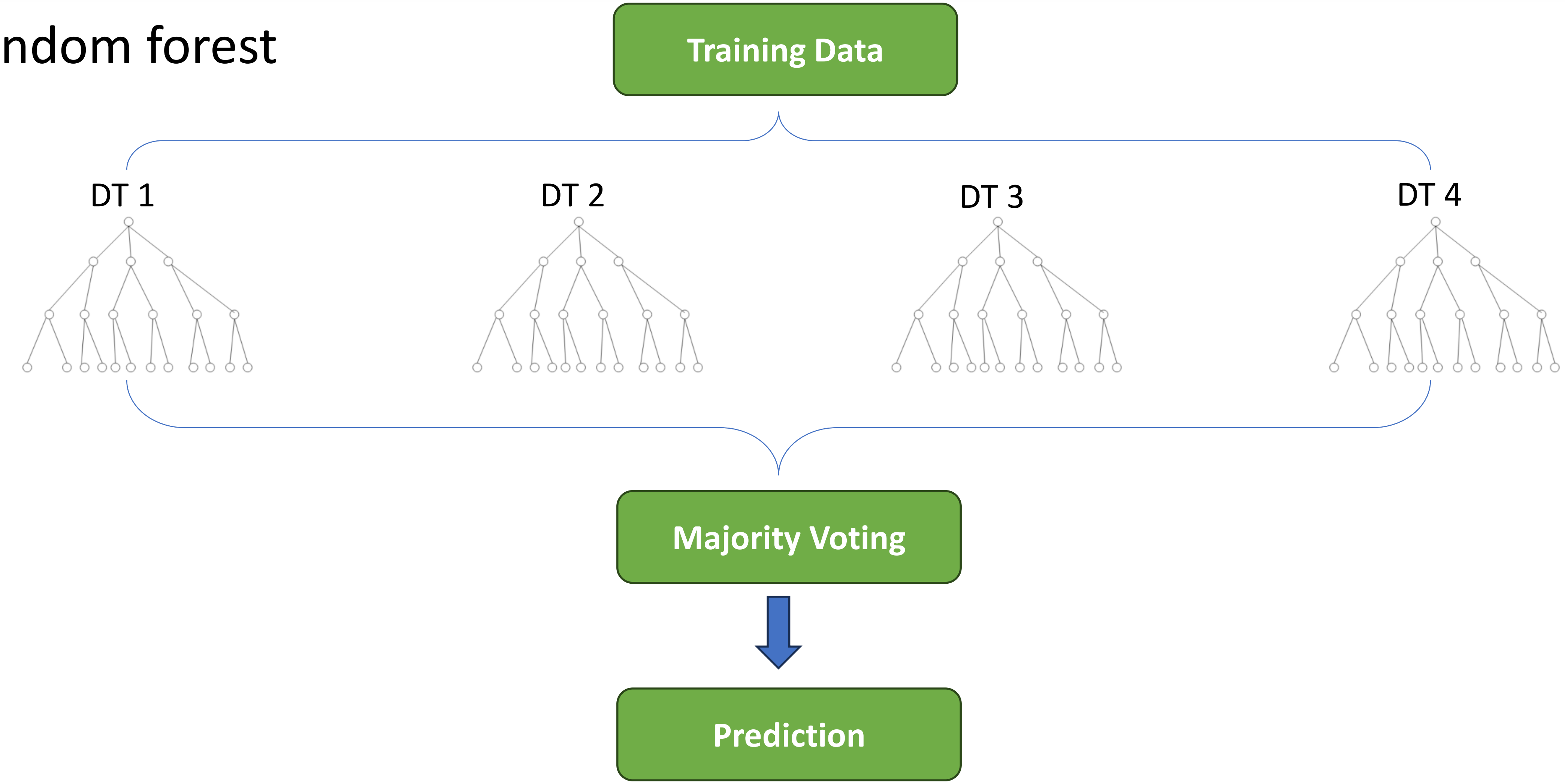
A two-stage approach

- Random forest
  - It works by creating a number of decision trees during the training phase.
  - Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition.
  - This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.
  - In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks).
  - This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results.

https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

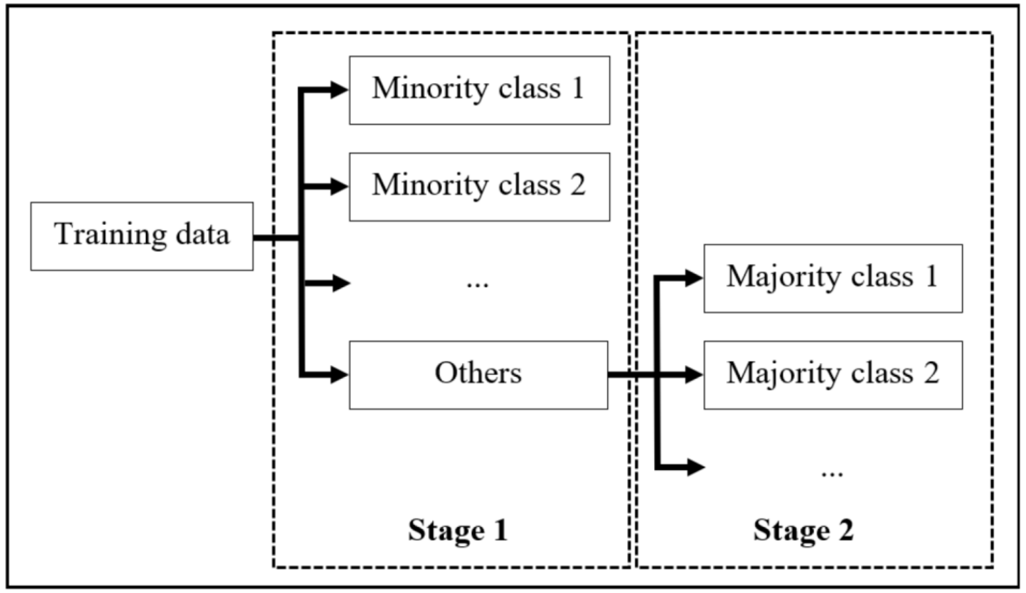
A two-stage approach

Random forest



A two-stage approach

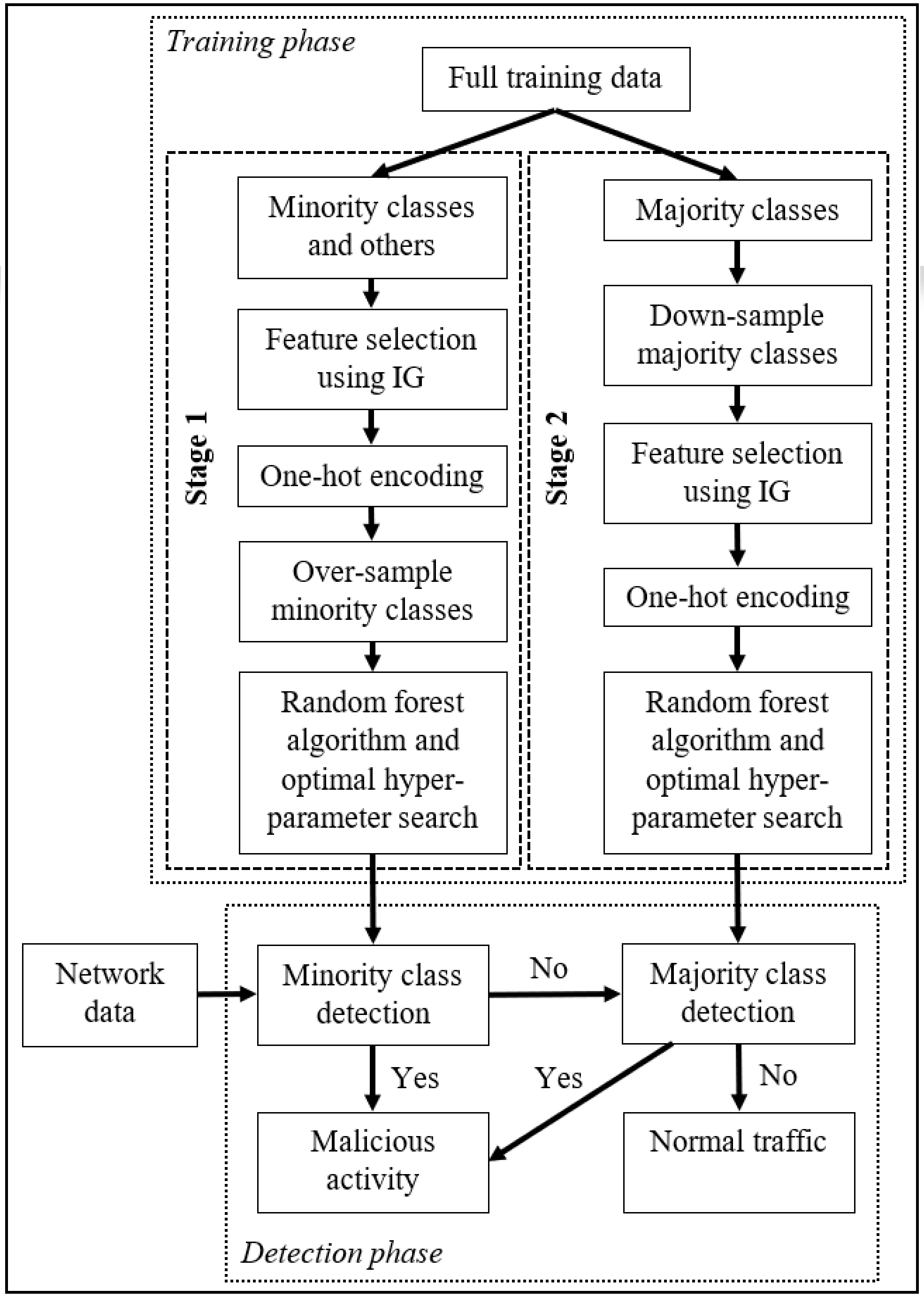
- Overview of the two-stage approach
  - Divide the dataset into majority and minority classes for the two stages, depicted in the figure.
  - During the **first stage**, majority classes, which occupy a major proportion of a training set, are classified as “others"
    - A classier is trained to identify the minority classes.
  - In the **second stage**, the minority classes are removed and another model is trained to identify the majority classes.
  - This results in two different intrusion detection models for identifying the minority and majority classes respectively.
    - It is possible to use different classifiers for each of the stages.



Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In *Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14* (pp. 329-340). Springer International Publishing.

A two-stage approach

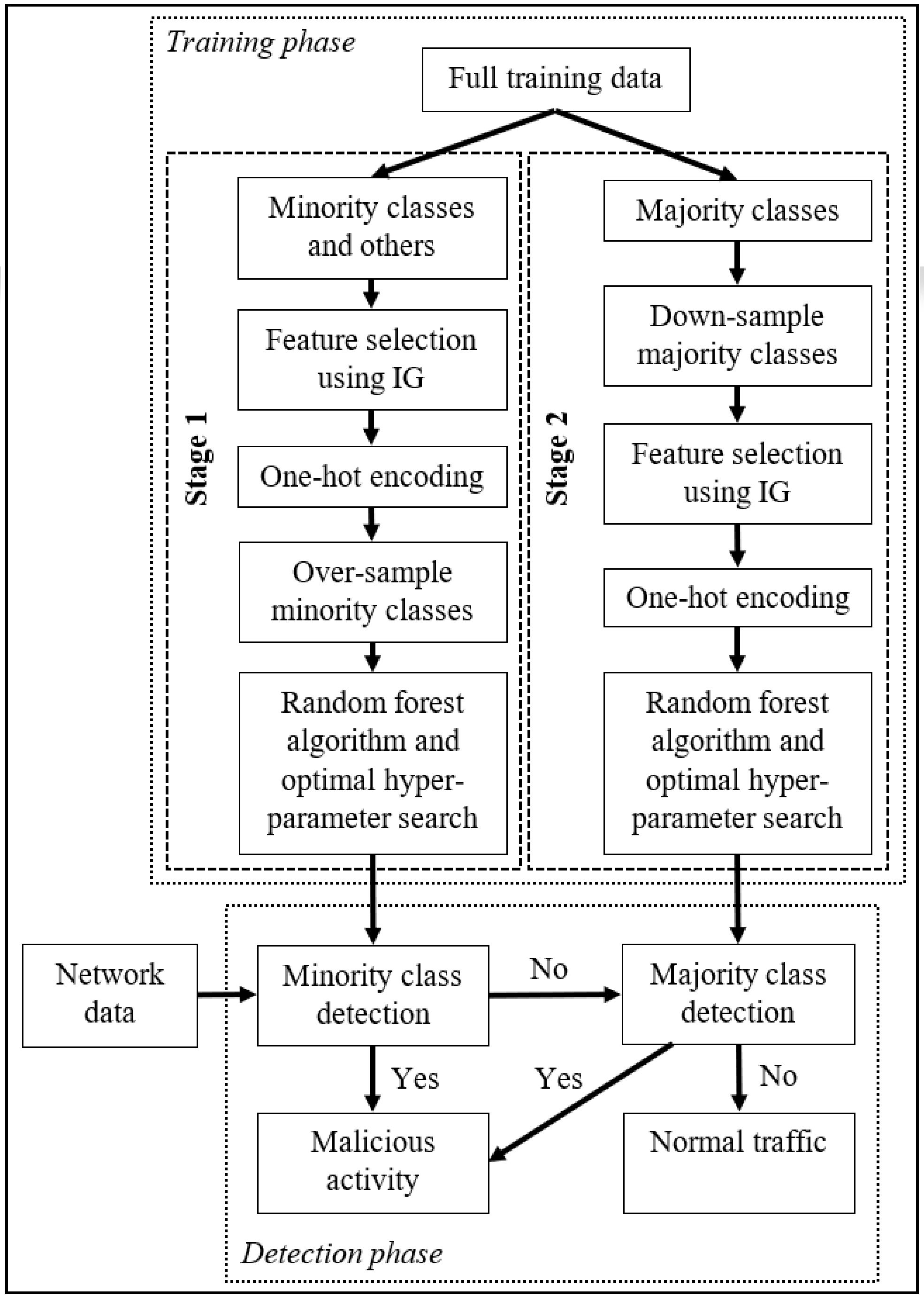
- **Stage 1 training phase**
  - After extracting the minority classes, feature selection is performed using the information gain method.
  - all categorical features are then converted into binary features using one-hot encoding to produce a set of numeric values.
  - The SMOTE method is then used to over-sample the minority classes.
  - The resulting set is then used for the training.
- **Stage 2 training phase**
  - Only the majority classes are used in the training set.
    - the minority classes are removed, since this was handled in stage 1.
  - Down-sampling is performed to balance the majority classes using a random selection method.
    - Network traffic within the majority classes in itself is still unbalanced, hence, down-sampling is performed to balance categories.
  - Information gain is again used for feature selection, followed by one-hot encoding.
  - The resulting set is then used for the training.



Training phase
Full training data
Stage 1
Minority classes and others
Feature selection using IG
One-hot encoding
Over-sample minority classes
Random forest algorithm and optimal hyper-parameter search
Stage 2
Majority classes
Down-sample majority classes
Feature selection using IG
One-hot encoding
Random forest algorithm and optimal hyper-parameter search
Network data
Minority class detection
No
Majority class detection
Yes
Yes
No
Malicious activity
Normal traffic
Detection phase

A two-stage approach

- Detection Phase
  - Network data is input into the system.
  - The model for identifying minority classes is used first to determine whether an activity is malicious.
  - If it is not identified as one of the minority classes, the second model is then applied to identify whether the activity is a majority intrusion.
  - Otherwise, it is determined to be normal network traffic.


Training phase
Full training data
Stage 1
Minority classes and others
Feature selection using IG
One-hot encoding
Over-sample minority classes
Random forest algorithm and optimal hyper-parameter search
Stage 2
Majority classes
Down-sample majority classes
Feature selection using IG
One-hot encoding
Random forest algorithm and optimal hyper-parameter search
Network data
Minority class detection
No
Majority class detection
Yes
Yes
No
Malicious activity
Normal traffic
Detection phase

A two-stage approach

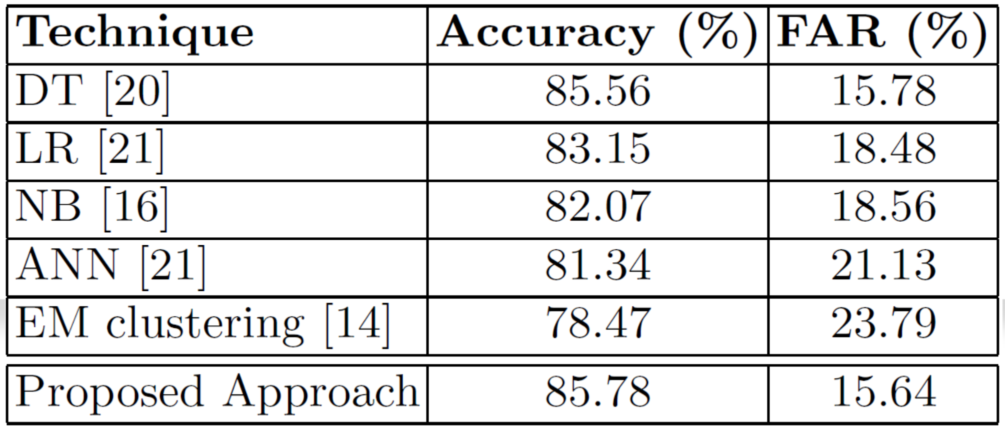
- UNSW-NB15 dataset
  - NSL_KDD is a benchmark datasets for intrusion detection
    - A classic dataset for evaluating models.
    - However, data were generated more than a decade ago, and these datasets arguably no longer reflect the current network threat environment.
  - the UNSW-NB15 dataset was proposed as a contemporary dataset
    - It was created as a hybrid of modern normal and contemporary synthesized attack activities of network Traffic.
      - 100 GB of the raw traffic was captured to build this dataset.
    - This dataset has nine types of attacks:
      - namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms.
    - 49 features for each record.
      - 41 features in NSL_KDD

A two-stage approach

- UNSW-NB15 dataset
  - The table shows the characteristics of the UNSW-NB15 dataset
    - where the training and testing sets have been divided into an approximately 60% to 40% ratio.
    - There were no redundant records among the training and testing set.
  - Different categories are unequally represented in the dataset.
    - For example, the analysis, backdoor, shellcode and worms categories are minority classes that collectively only make up < 3% of the sets.
    - This imbalance creates problems for classifiers and results in poor detection performance of these minority classes.
  - The majority classes like normal, exploits, generic, etc. occur frequently and there is an abundance of such training samples.

Category	**Training Set**	**Testing Set**
Normal	56,000	37,000
Analysis	2,000	677
Backdoor	1,746	583
DoS	12,264	4,089
Exploits	33,393	11,132
Fuzzers	18,184	6,062
Generic	40,000	18,871
Reconnaissance	10,491	3,496
Shellcode	1,133	378
Worms	130	44
Total Records	175,341	82,332

A two-stage approach



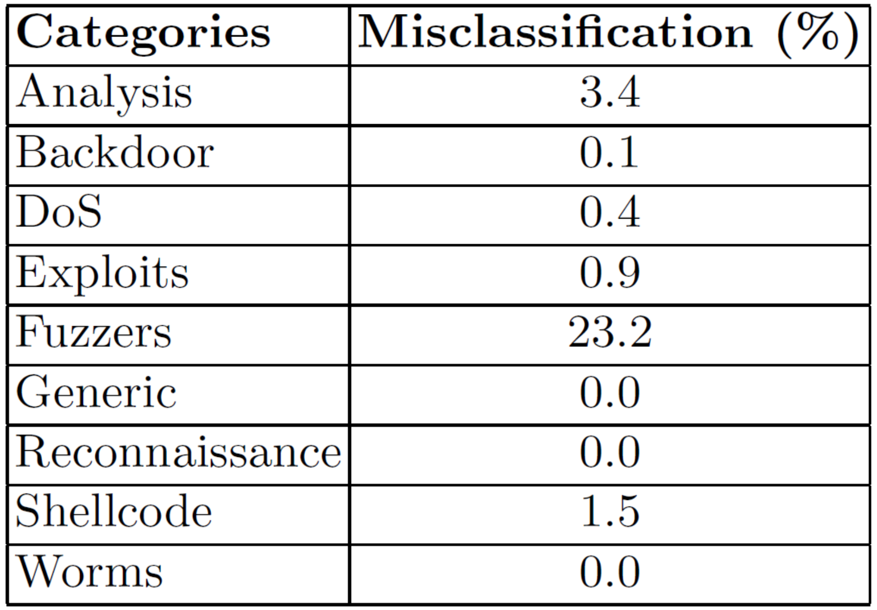
Technique	Accuracy (%)	FAR (%)
DT [20]	85.56	15.78
LR [21]	83.15	18.48
NB [16]	82.07	18.56
ANN [21]	81.34	21.13
EM clustering [14]	78.47	23.79
Proposed Approach	85.78	15.64

- Detection results
  - A comparison with the Decision Tree (DT), Logistic Regression (LR), Naïve Bayes (NB), Artificial Neural Network (ANN) and Expectation-Maximization (EM) clustering)
  - The resulting accuracy of the proposed approach is higher than the other techniques, while the False Alarm Rate (FAR) is lower.
    - Overall results are comparable with the results of DT.
      - Random forest is based on decision trees.
      - DT would miss minority classes.
      - Other types of classifiers can potentially be used to further improve results.
      - The two-stage framework sheds light on more advanced methods.

A two-stage approach

- Analysis of misclassification
  - The table provides the rate of normal traffic that was misclassified as malicious activity.
  - Most of the misclassification was to do with fuzzing activity.
    - It can be seen from the table at a large portion of the misclassication are for fuzzers.
  - Fuzzers are attacks where the attacker attempts to discover security loopholes in a program, operating system or network.
    - They are not necessarily dangerous in themselves when compared with other attacks.
    - Fuzzing activity has to do with inputting lots of random data.
    - As such, they do not have a fixed pattern and are more difficult to distinguish from normal network traffic.

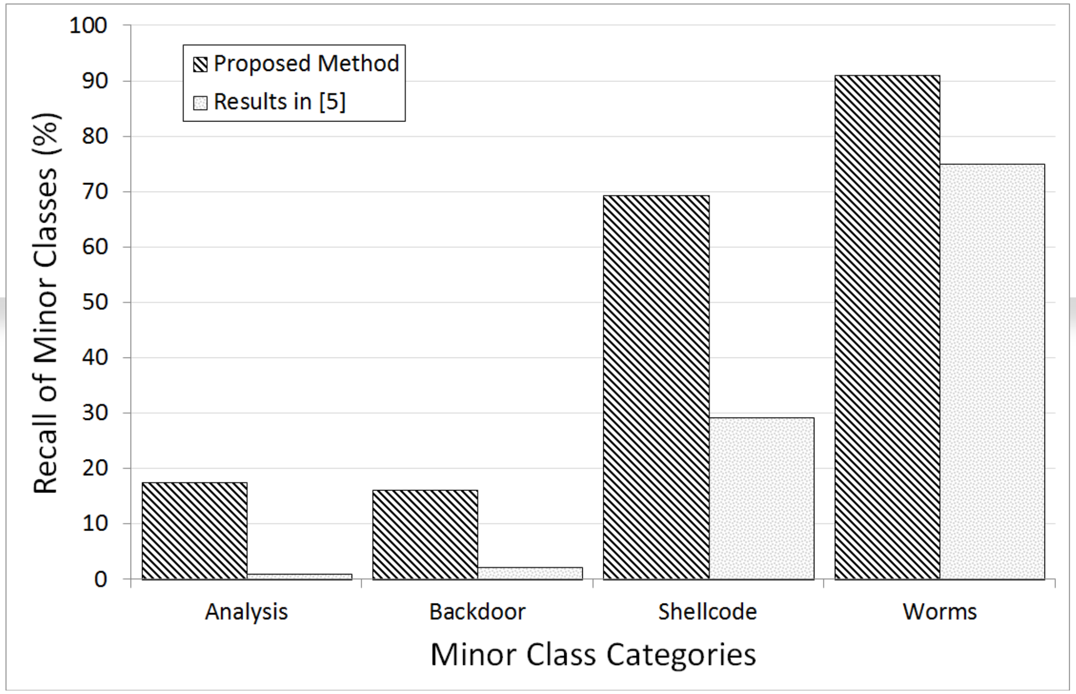
Normal activity misclassified as malicious.

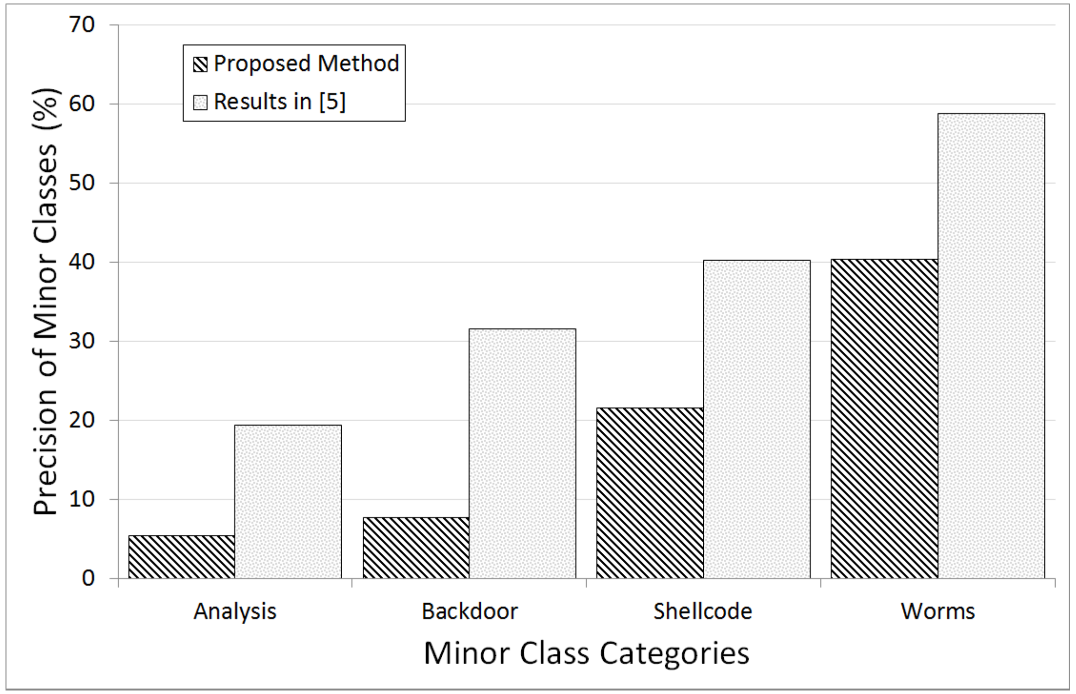


Categories	Misclassification (%)
Analysis	3.4
Backdoor	0.1
DoS	0.4
Exploits	0.9
Fuzzers	23.2
Generic	0.0
Reconnaissance	0.0
Shellcode	1.5
Worms	0.0

A two-stage approach

- Minor classes recall vs precision
- The 2 figures provide a comparison of the recall and precision performances, respectively
  - between this two-stage random forest (“Proposed Method”) and single-stage random forest (“Results in [5]”).
- The recall results of the this two-stage approach performs better in comparison.
  - However, the precision performance is lower.
- Nevertheless, for minority classes a higher recall rate is more important than precision
  - Because these attacks are potentially more dangerous than other attacks.
  - Therefore, higher recall values prevent these attacks from escaping detection.





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.
  - 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.
  - The improvement of ML models usually relies on a trial-and-error process.
    - Due to the complex nature of ML mechanisms.
  - 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.

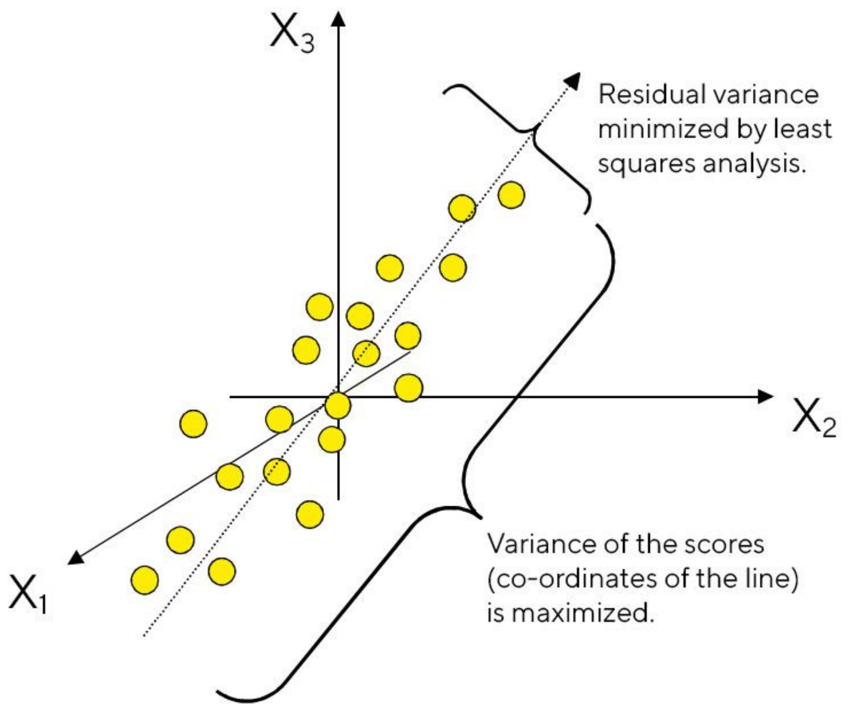
3D Visualization of Network Intrusion Detection Data

- 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.

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.

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.


X_3
Residual variance minimized by least squares analysis.
X_2
X_1
Variance of the scores (co-ordinates of the line) is maximized.

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

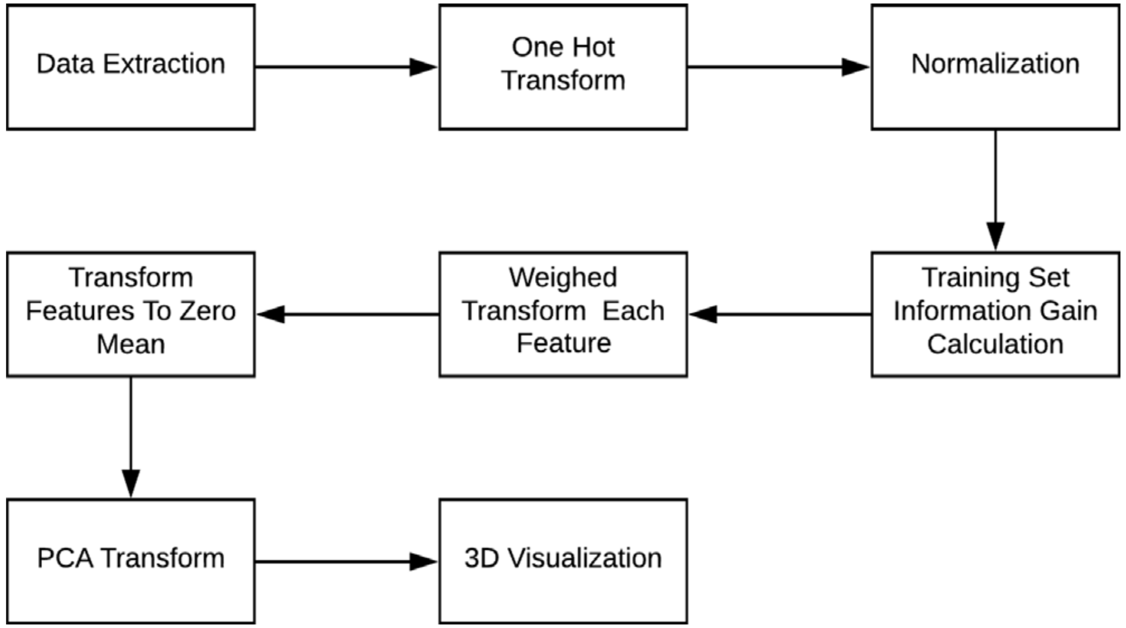
3D Visualization of Network Intrusion Detection Data

- PCA vs t-SNE
  - t-distributed stochastic neighbor embedding (t-SNE) is also widely used for visualization.
  - The adopted dimensionality reduction technique must fulfill the following requirements:
    - First, it should preserve the geometric relationship between the network traffic records, since this relationship is important for ML models, like the SVM technique, in producing the ML decision spaces for the proposed approach,.
    - Second, it should learn a parametric mapping so that new data can be mapped into the low dimensional space.
      - Otherwise, it cannot handle new data.
  - Although t-SNE usually provides better data visualization results than PCA. it does not satisfy the requirements mentioned above.
    - t-SNE does not retain geometric relationship between data,
    - t-SNE Does not learn a parametric mapping and the transformation to low dimension space.
  - In contrast, the PCA technique fulfills all these requirements.

https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186#:~:text=Principal%20component%20analysis%2C%20or%20PCA,more%20easily%20visualized%20and%20analyzed.

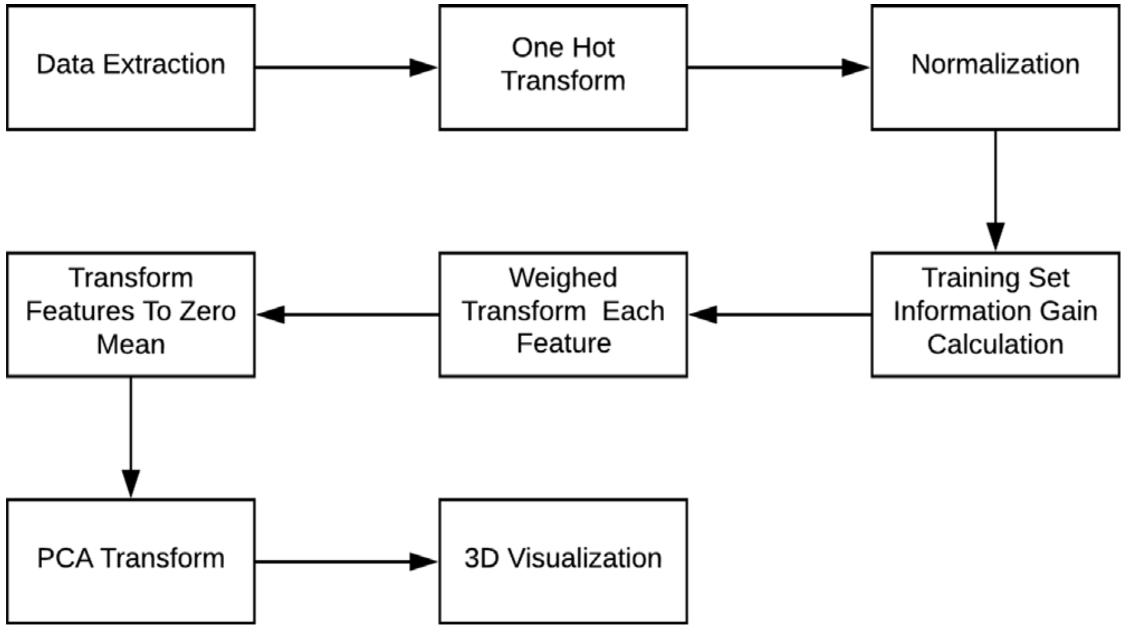
3D Visualization of Network Intrusion Detection Data

- Step 1/8
  - All the data for the minor categories are extracted from the training set.
    - Minor categories like worm attacks (in the UNSW-NB15 dataset), and U2R and R2L attacks (in the NSL_KDD dataset), only occupy a small portion of the dataset.
  - Major categories data are randomly extracted until a certain predefined amount, e.g., 30% of the data.
    - The random sampling of the major categories does not diminish the quality of visual information,
    - It is useful to reduce the visual clutter and the amount of required computation on the visualization system.
  - Extraction is not performed on the testing set
    - All the test data is used.


Data Extraction
One Hot Transform
Normalization
Training Set Information Gain Calculation
Weighed Transform Each Feature
Transform Features To Zero Mean
PCA Transform
3D Visualization

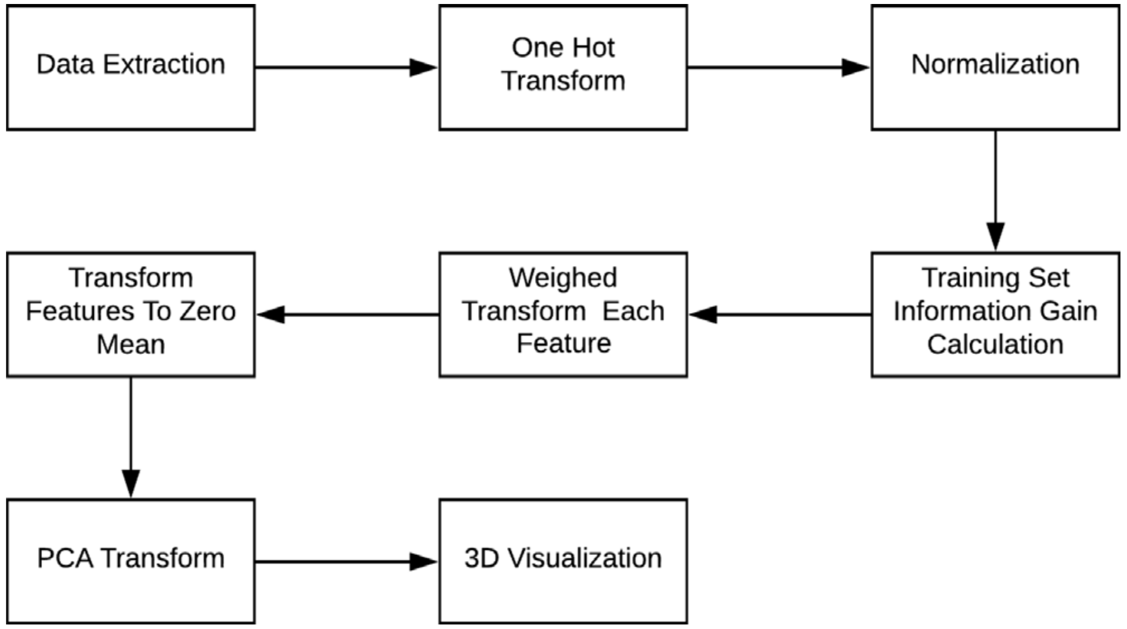
3D Visualization of Network Intrusion Detection Data

- Step 2/8
  - Apply one-hot transformation to categorial features.
- Step 3/8
  - Normalize the range of the features.
    - the range of values for the features can differ significantly. Some values may range between zero to less than a hundred, while others may range from 0 to several millions.
    - Linear normalization is applied to each feature in both the training and testing sets to normalize the values to the minimum and maximum range of the training set.


Data Extraction
One Hot Transform
Normalization
Training Set Information Gain Calculation
Weighed Transform Each Feature
Transform Features To Zero Mean
PCA Transform
3D Visualization

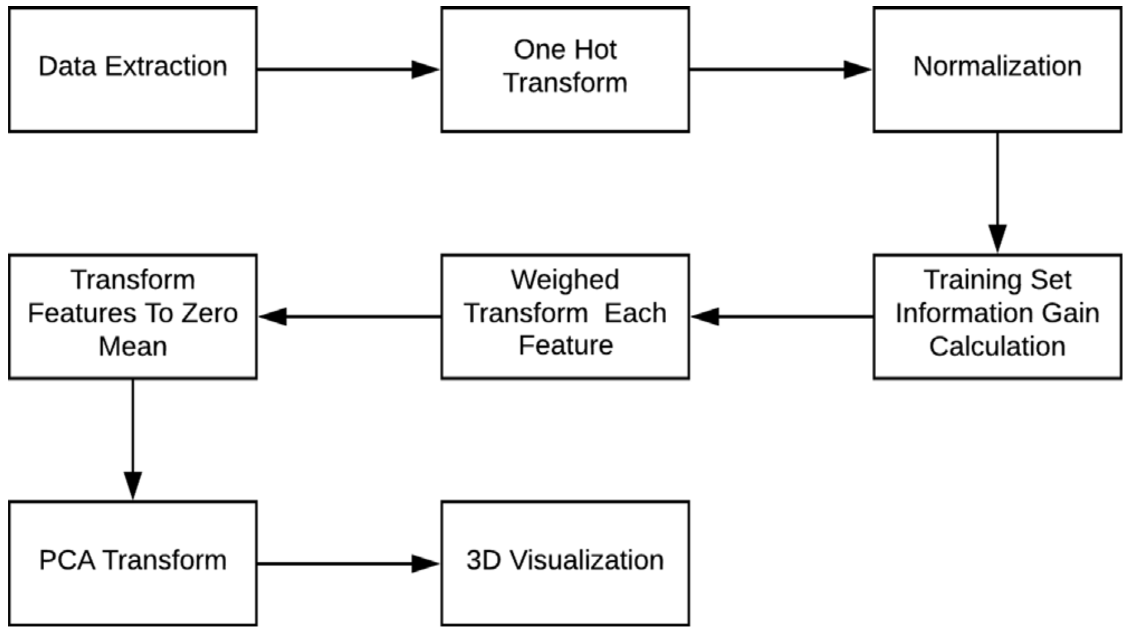
3D Visualization of Network Intrusion Detection Data

- Step 4/8
  - Calculate the information gain (IG) values for each feature.
- Step 5/8
  - IG values are normalized between 0 and 1, and weighted transform is applied by multiplying each feature with the corresponding information gain value.
    - As a result, the variances of important features either remain or are decreased slightly, whereas the variances of unimportant features are significantly reduced.


Data Extraction
One Hot Transform
Normalization
Training Set Information Gain Calculation
Weighed Transform Each Feature
Transform Features To Zero Mean
PCA Transform
3D Visualization

3D Visualization of Network Intrusion Detection Data

- Step 6/8
  - All features are transformed to zero mean in the training set.
    - The testing set is transformed based on mean values of the corresponding features from the training set.
- Step 7/8
  - Apply PCA.
- Step 8/8
  - Visualize the first 3 components (Unity Game Engine).

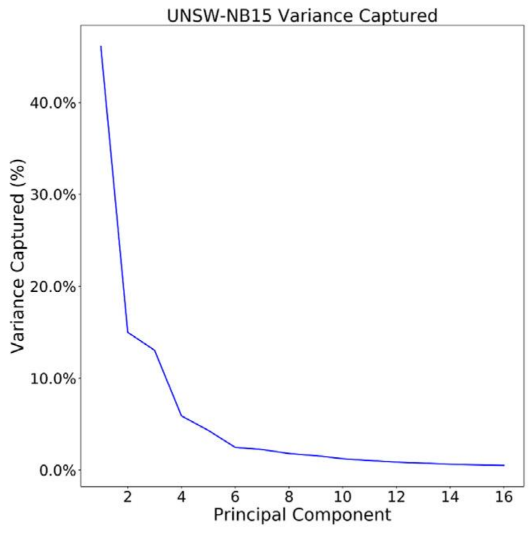

Data Extraction
One Hot Transform
Normalization
Training Set Information Gain Calculation
Weighed Transform Each Feature
Transform Features To Zero Mean
PCA Transform
3D Visualization

3D Visualization of Network Intrusion Detection Data

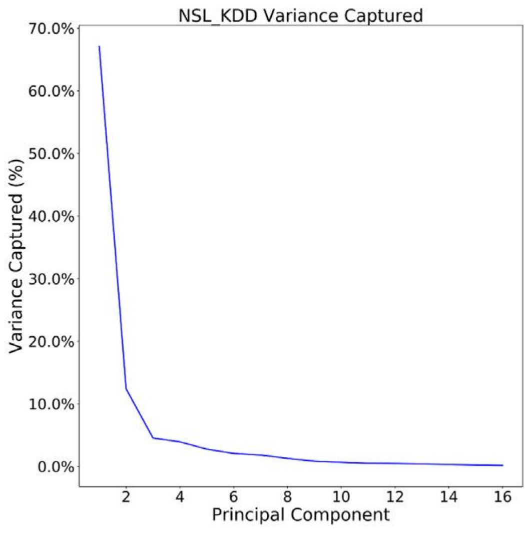
- 3D or 2D visualization?

3D Visualization of Network Intrusion Detection Data

- 3D or 2D visualization?
  - PCA replaces original variables with new variables, called **principal components**, which are orthogonal.
  - Principal components have variances in a descending order.
    - The amount of variance by each of the selected components is referred to as “variance explained”.
    - Indicate how well data can be described in low dimensional space.
  - The third component captures 13.0% and 4.5% variances for UNSW-NB15 and NSL_KDD datasets respectively
    - 3D visualization is better than 2D visualization
    - Three components obviously capture more variance in both datasets than only two components.



(a)



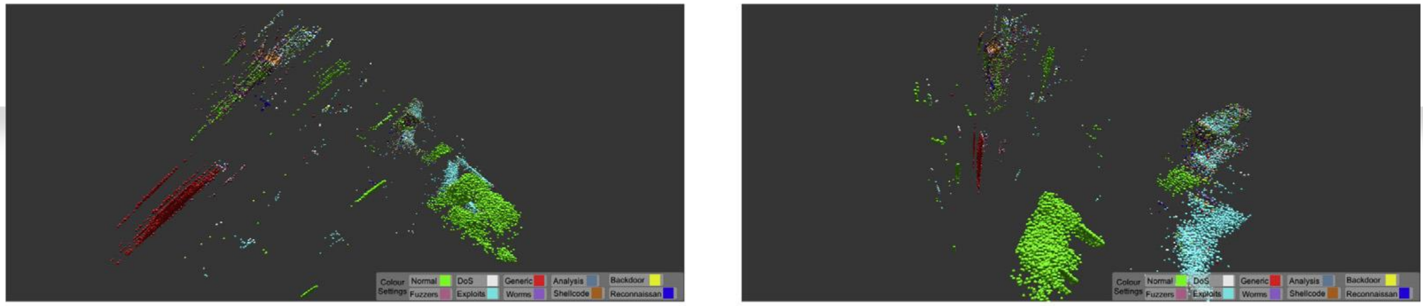
(b)

3D Visualization of Network Intrusion Detection Data

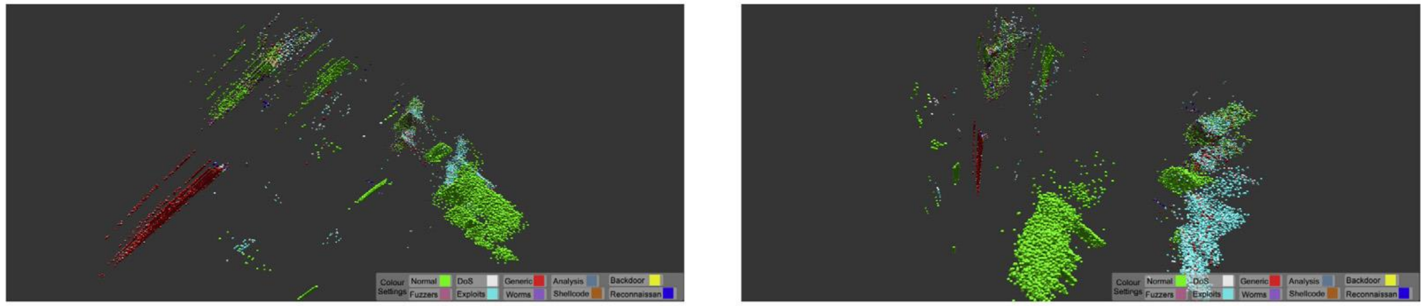
- Visually explore NIDS datasets (SVM was the classifier):
  - KDD_No_Boundaries_All_Data.mp4
  - KDD_Binary_Overall.mp4
  - KDD_Multi_Overall.mp4
  - UNSW_No_Boundaries_All_Data.mp4
  - UNSW_Binary_overall.mp4
  - UNSW_Multi_overall.mp4

3D Visualization of Network Intrusion Detection Data

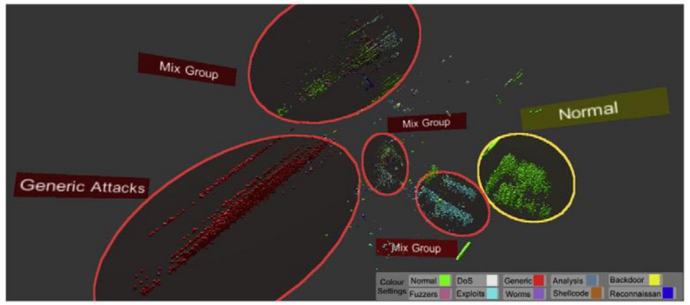
- Visually explore **UNSW-NB15**
  - **(a)** shows a visual depiction of a portion of the training set from two different camera viewpoints, where the different network traffic categories have been color coded to visually distinguish them from one another.
    - It can be seen visually that traffic from 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.
  - **(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)



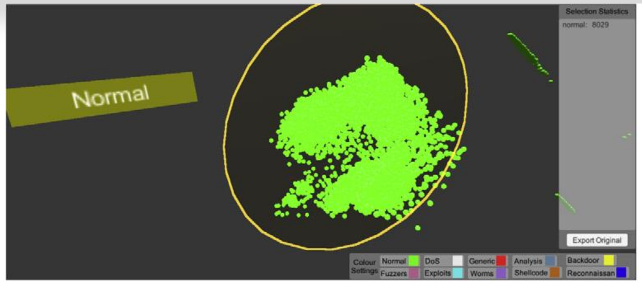
(b)



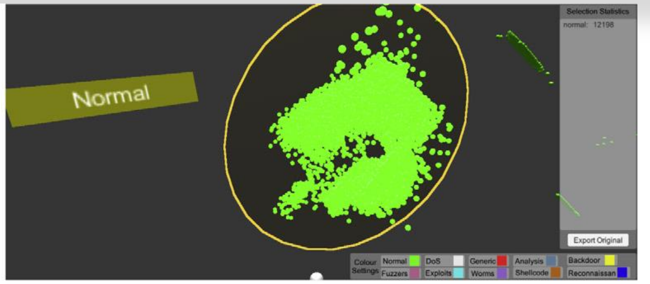
(c)

3D Visualization of Network Intrusion Detection Data

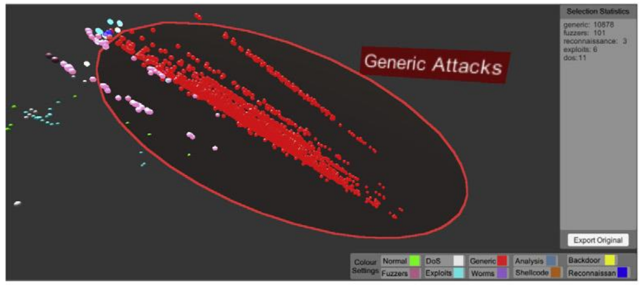
- A closer investigation of **UNSW-NB15**
  - Examples of normal traffic clusters from the training and testing data are shown in (a) and (b), respectively.
    - It can be seen from the figures that the characteristics of the training and testing data are very similar.
    - The implication of this 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.
    - The two-stage approach for generic attacks :
      - recall 96.6%, precision 99.9%



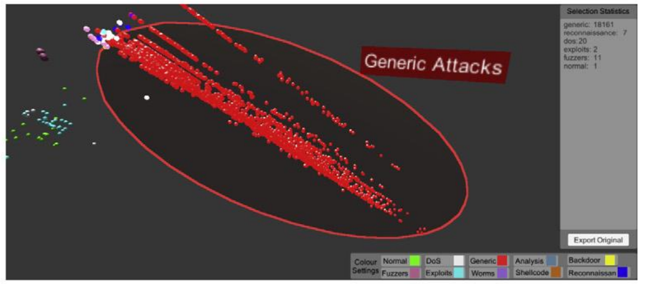
(a)



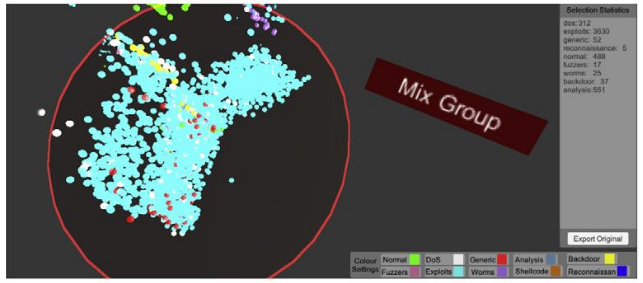
(b)



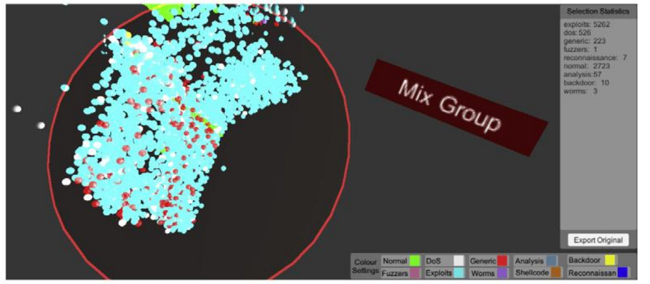
(c)



(d)



(e)

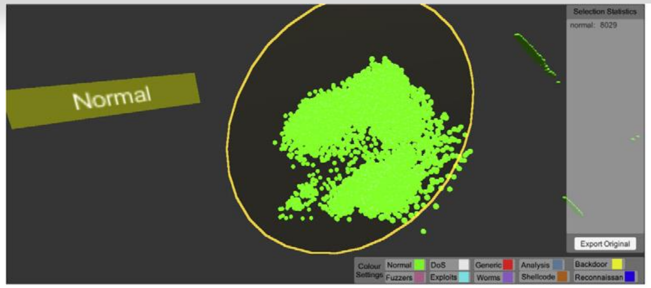


(f)

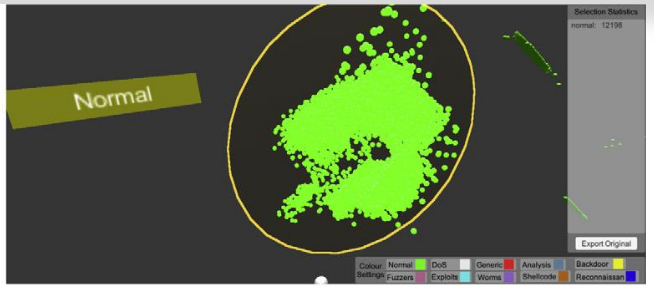
Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

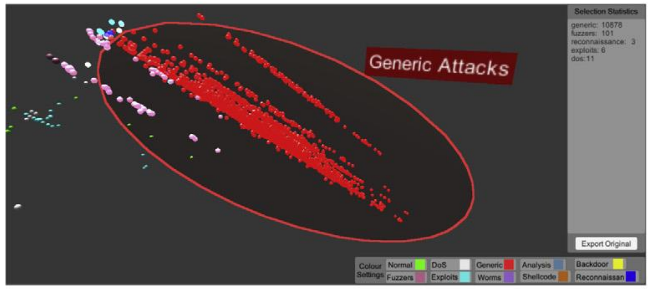
- A closer investigation of **UNSW-NB15**
  - In other sections of the dataset, the traffic is mixed together and there is no clear visual distinction between the different traffic categories within these clusters.
  - (e) and (f) for the training and testing data, respectively.
    - Such sections of 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.



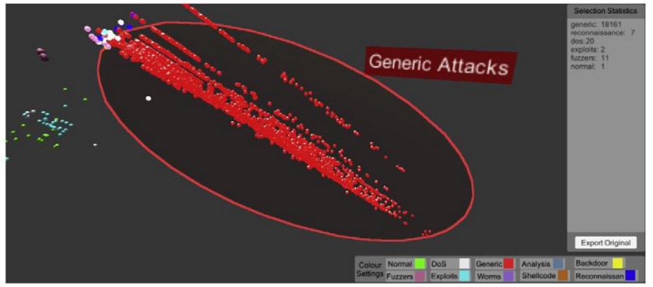
(a)



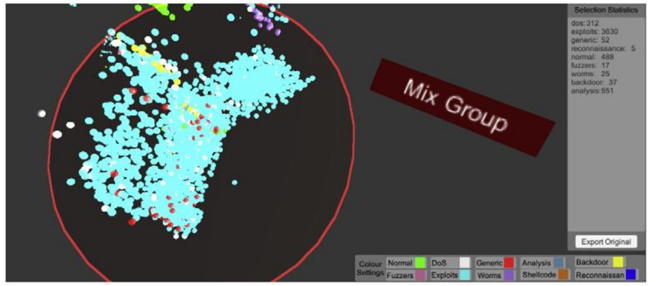
(b)



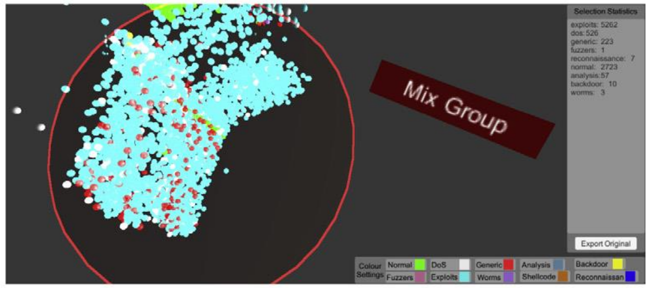
(c)



(d)



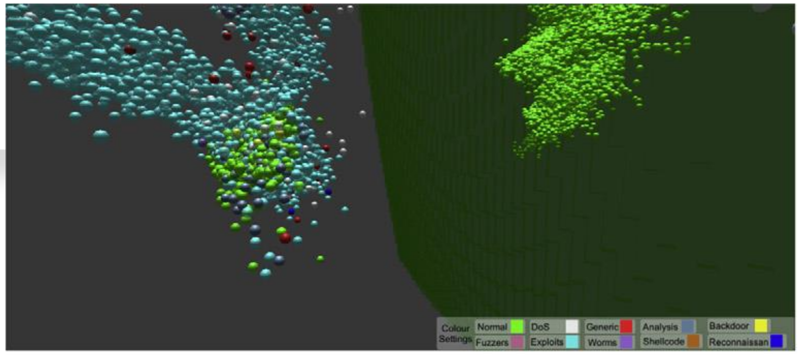
(e)



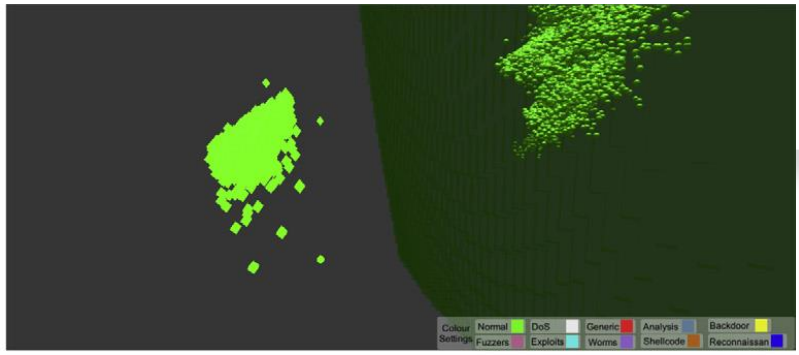
(f)

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.

3D Visualization of Network Intrusion Detection Data



(a)

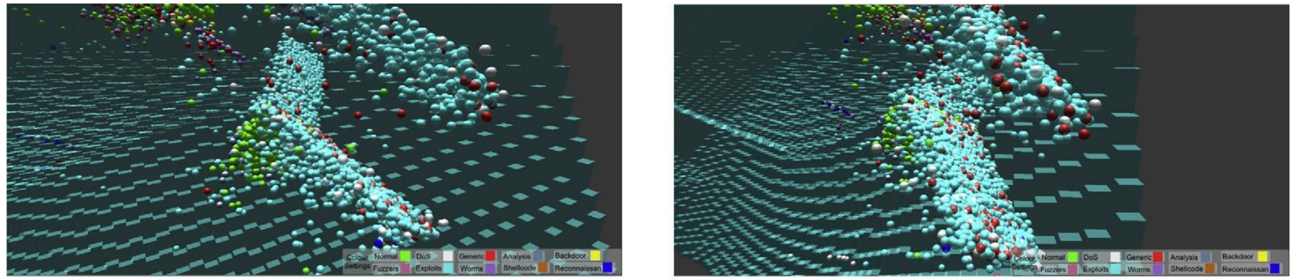


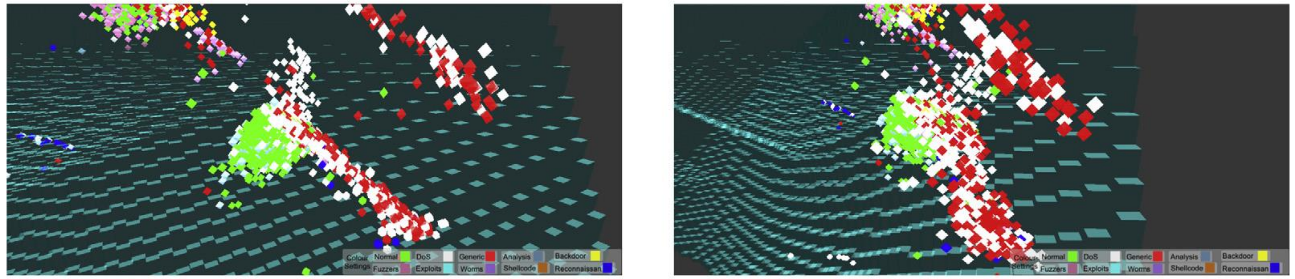
(b)

- 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

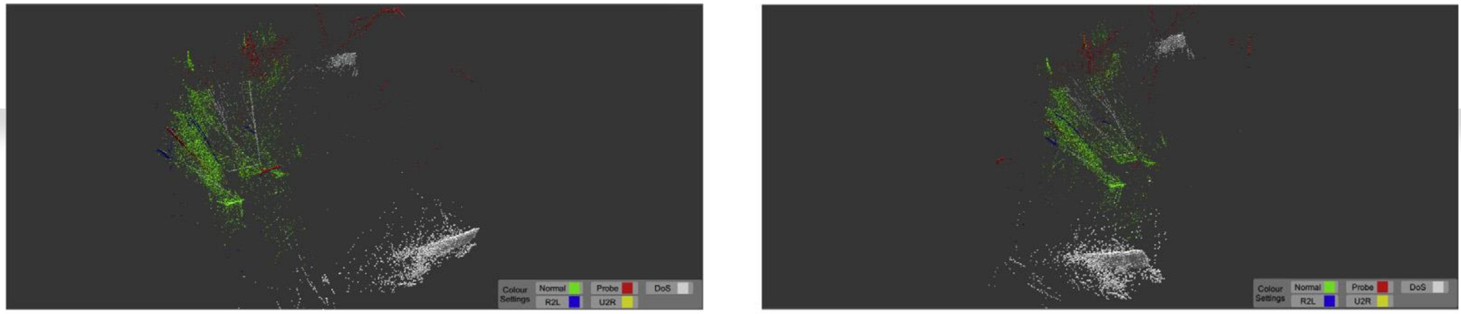
- A closer investigation of **UNSW-NB15**
  - In some clusters, in which there was a greater proportion of exploit attacks than other categories, other traffic would potentially be misclassified as exploits.
  - This was verified in the results, as can be seen in the example shown in the figures.
    - 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)

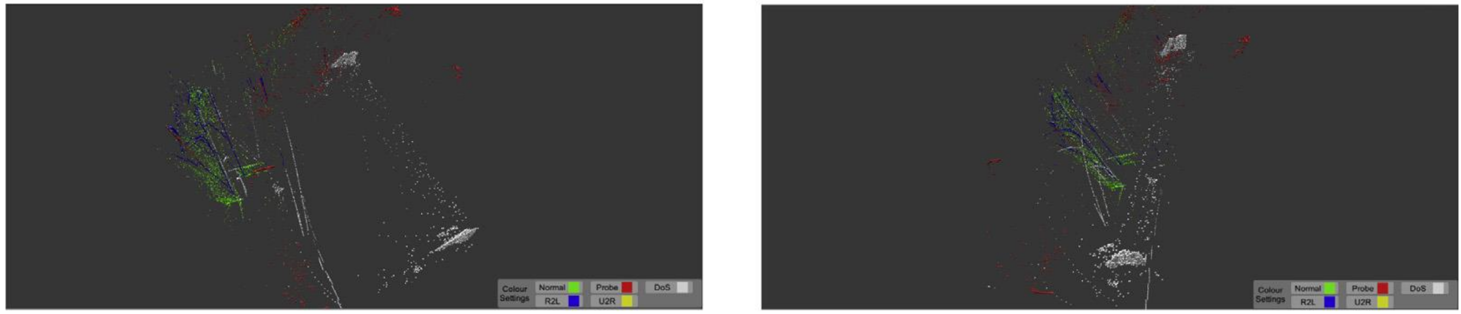

(b)

3D Visualization of Network Intrusion Detection Data

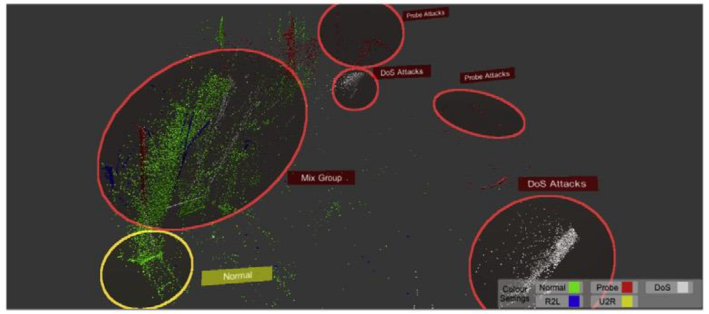
- Visually explore **NSL_KDD**
  - Figures show the overall 3D visual representation of the NSL_KDD dataset.
    - Figure (a) depicts a portion of the data from the training set,
    - Figure (b) displays part of the data from the testing set, from two different camera viewpoints respectively.
  - While 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)



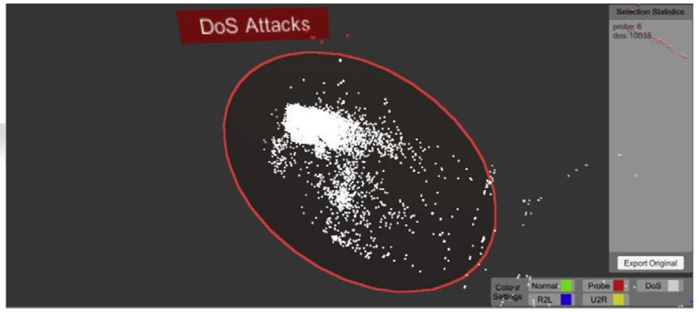
(b)



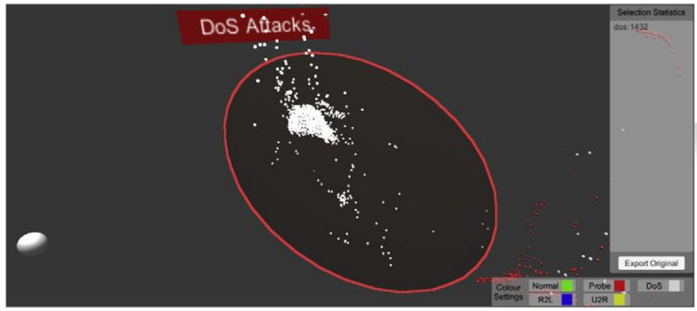
(c)

3D Visualization of Network Intrusion Detection Data

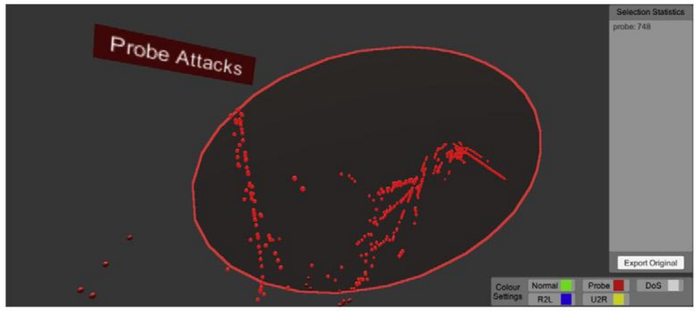
- A closer investigation of **NSL_KDD**
  - From the visual representation, one can observe that the data consists of clusters that contain mainly homogeneous records as well as clusters that contain diverse traffic.
  - Figures (a) and (b) show examples of DoS attack traffic from the training and testing 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.



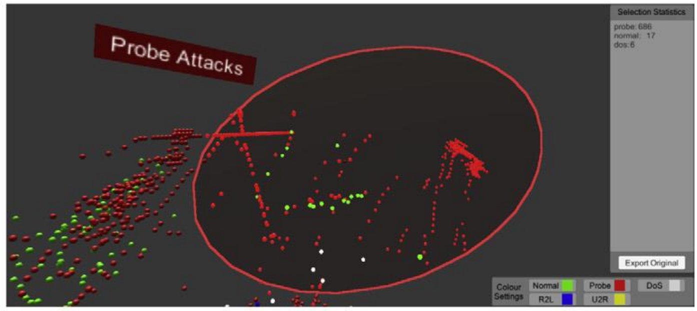
(a)



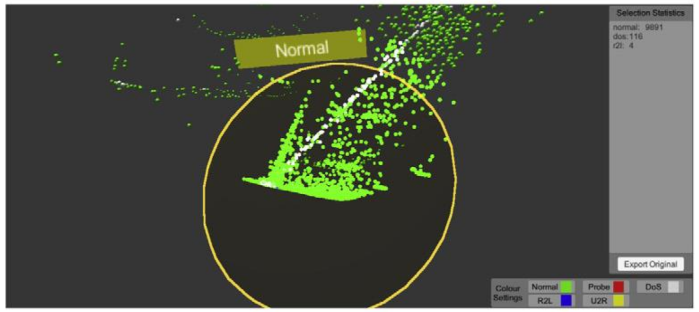
(b)



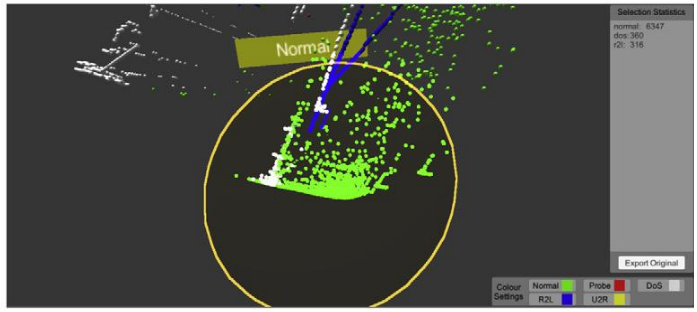
(c)



(d)



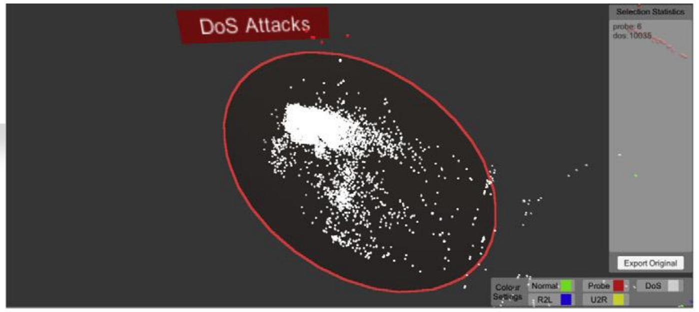
(e)



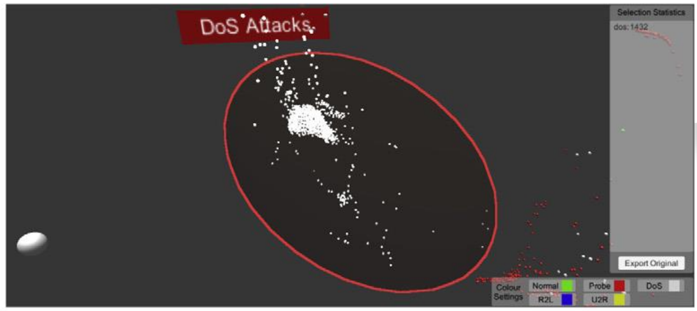
(f)

3D Visualization of Network Intrusion Detection Data

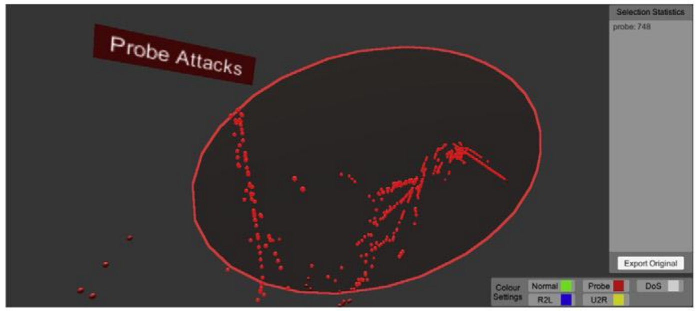
- A closer investigation of **NSL_KDD**
  - Figures (c) and (d) show another example of a cluster containing distinguishable homogeneous traffic
    - containing probe attacks, as from the training and testing data, respectively.
  - Although in (d) there are other traffic near the cluster, such as normal 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.



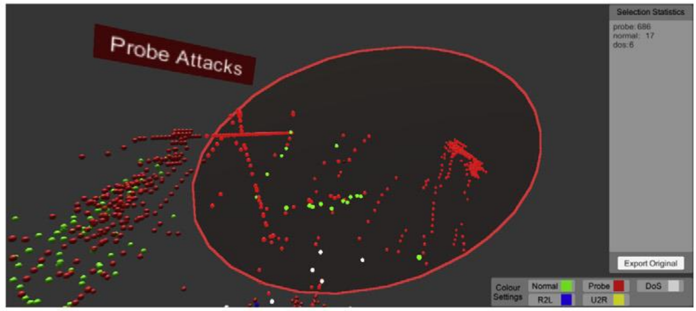
(a)



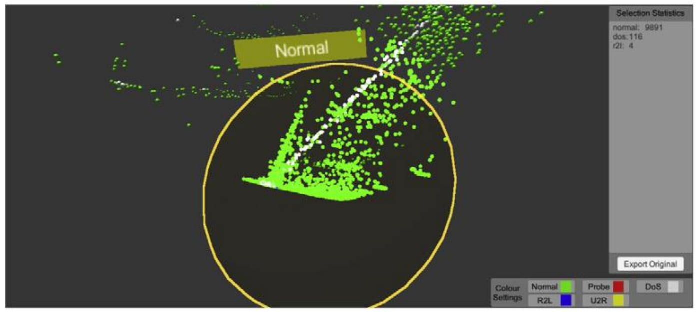
(b)



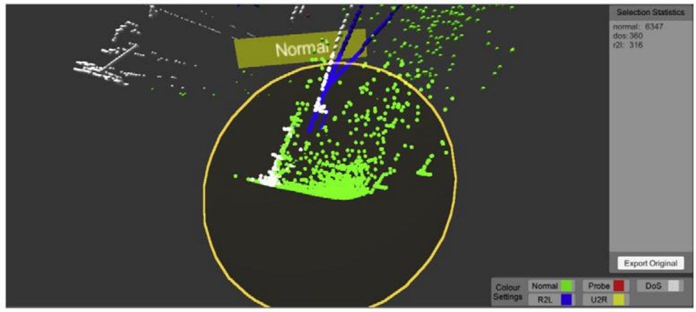
(c)



(d)



(e)

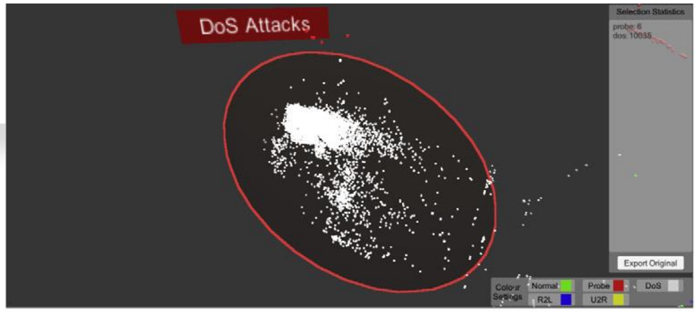


(f)

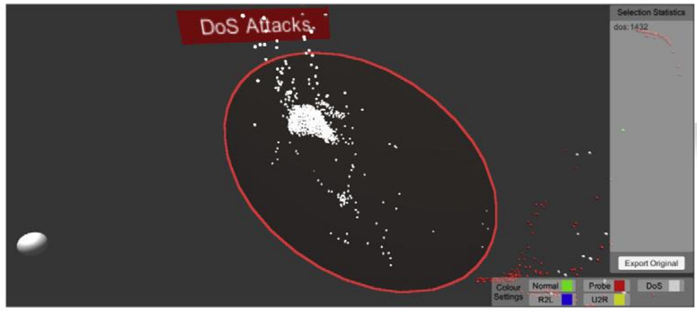
Zong, W., Chow, Y.W. and Susilo, W., 2020. Interactive three-dimensional visualization of network intrusion detection data for machine learning. *Future Generation Computer Systems*, *102*, pp.292-306.

3D Visualization of Network Intrusion Detection Data

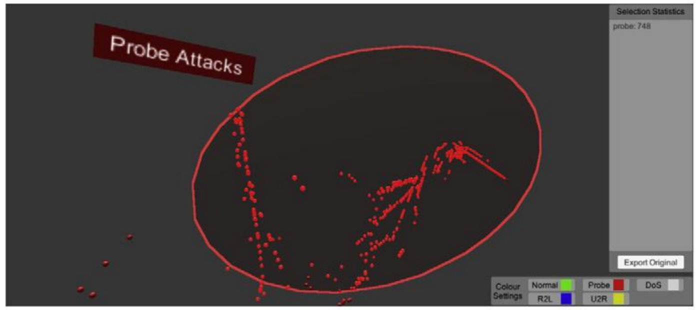
- A closer investigation of **NSL_KDD**
  - Cyber security experts have identified that the main difficulty faced by ML techniques in the NSL_KDD comes from previously unknown attacks in the testing set [1].
    - Even though the categories of attacks are the same in the training and testing sets, the 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.



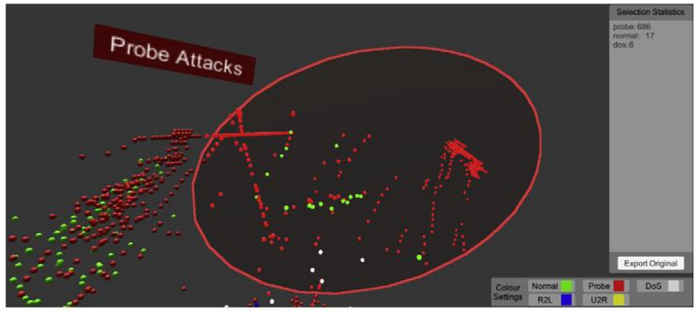
(a)



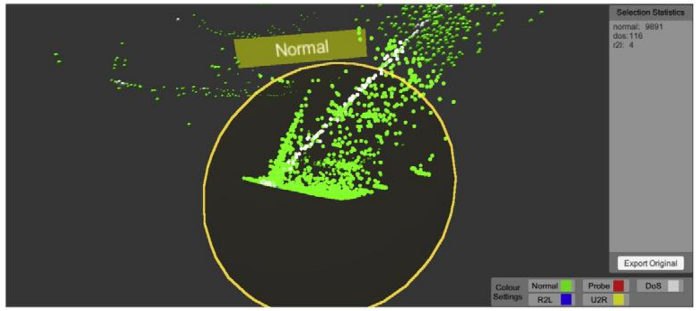
(b)



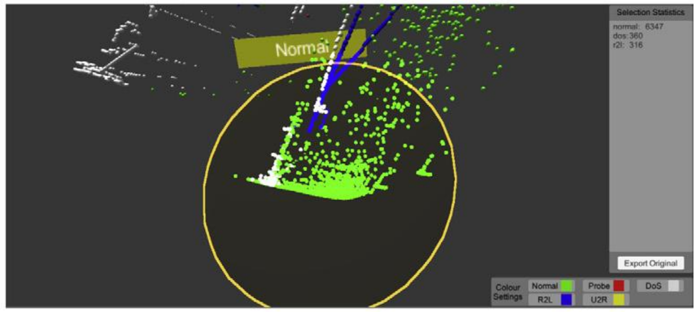
(c)



(d)



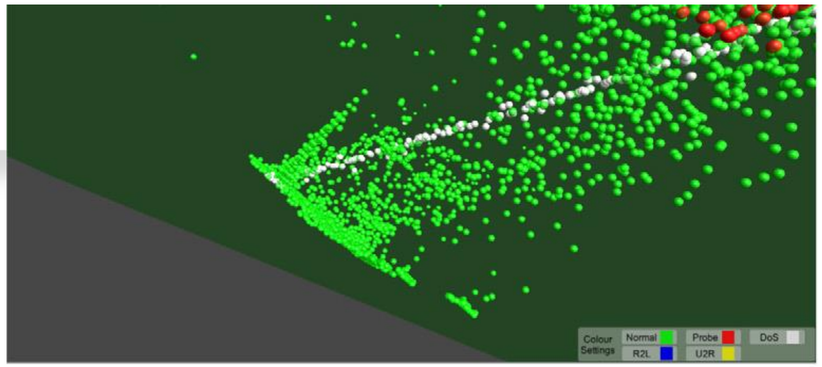
(e)



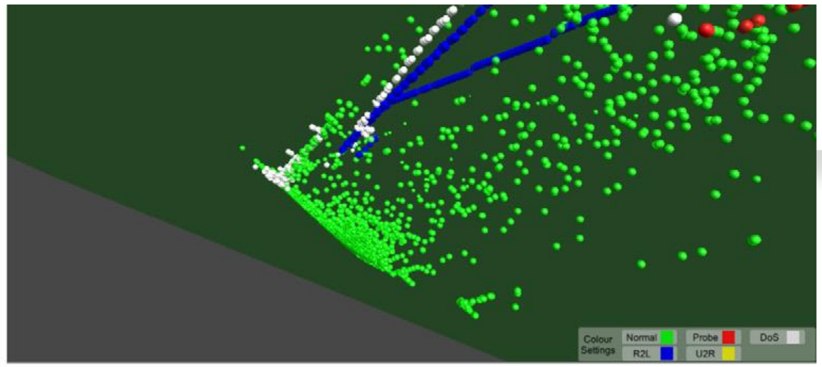
(f)

[1] 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.

3D Visualization of Network Intrusion Detection Data



(a)



(b)

- A closer investigation of **NSL_KDD**
  - Visualizing decision boundaries of an SVM model.
    - Since 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.

[1] 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.

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.

Conclusion

- Machine learning is about more than just choosing the right classifier and knowing about algorithms, iterative process
- Key: spend time to explore and understand the data
- Cybersecurity provides some unique challenges when applying machine learning, e.g. class imbalance, lack of training data…
  - Sometimes continue to tune algorithms or the learning process may not be the answer
- To obtain better results
  - Collecting more data
  - generating more descriptive features
  - …

References

- *[1] 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.*
- *[2] Network Traffic Analysis, Machine Learning and Security Protecting Systems with Data and Algorithms, Clarence Chio, David Freeman*
- *[3] Zong, W., Chow, Y.W. and Susilo, W., 2018. A two-stage classifier approach for network intrusion detection. In Information Security Practice and Experience: 14th International Conference, ISPEC 2018, Tokyo, Japan, September 25-27, 2018, Proceedings 14 (pp. 329-340). Springer International Publishing.*
- *[4] 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.*