Traffic feature importance

Roberto Doriguzzi Corin

Cybersecurity centre

Fondazione Bruno Kessler

Outline

- Traffic feature engineering
  - Packet-level features
  - Flow-level features
- Techniques to expose relative feature importance and select relevant features

Feature engineering

Process that involves:

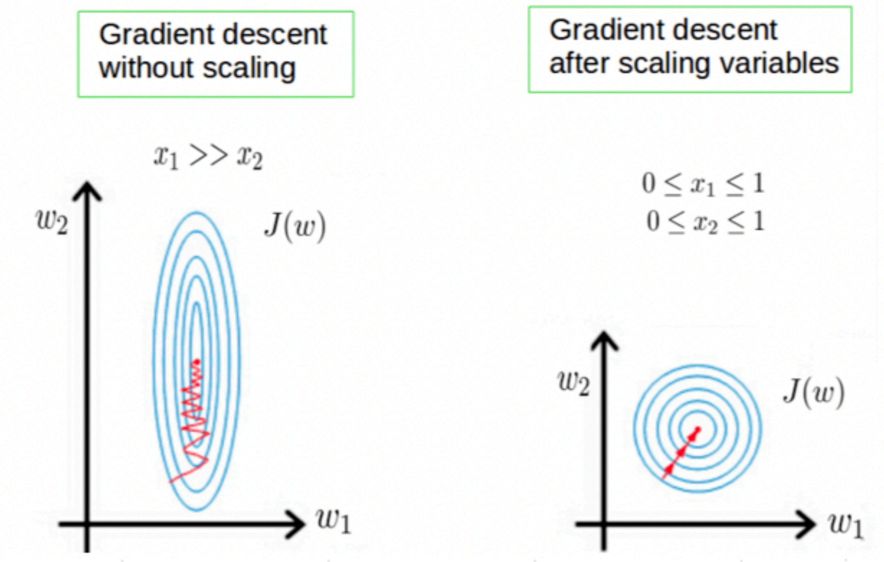
- **Feature selection** (selecting the most useful features)
- **Feature extraction**
  - extracting relevant features from the raw data
  - combining existing features to produce a more useful one
- **Feature processing**
  - Normalisation of the features into a common range (e.g., [0,1])
  - Encoding categorical data into integer format (e.g., one-hot encoding)

Feature processing (1)

Normalisation

- Neural networks **converge much faster** with normalised features
- Normalisation (or *min-max scaling*) shifts and re-scales all the features to [0,1] range

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$



Source: Stanford University

$$w_j := w_j - \alpha \cdot \frac{\partial}{\partial w_j} J(W)$$

$$:= w_j - \alpha \cdot \frac{\partial}{\partial w_j} \frac{1}{2m} \sum_{i=1}^{m} (h_W(x^{(i)}) - y^{(i)})^2$$

$$:= w_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_W(x^{(i)}) - y^{(i)}) \boxed{\cdot\, x_j^{(i)}}$$

The presence of feature value x_j in the formula affects the step size of the gradient descent

Feature processing (2)

One-hot-encoding

Categorical data are variables that contain label values rather than numeric values (e.g., 5 traffic classes: Benign, DDoS, Brute Force, Port Scan, SQL Injection).

Many machine learning algorithms cannot operate on text data directly. They require all input variables and output variables to be numeric.

Two steps:

- Integer encoding (e.g., Benign=0, DDoS=1, etc,)
- One-hot encoding (Benign=[1,0,0,0,0], DDoS=[0,1,0,0,0])

Feature extraction

Reminder: we are looking for malicious traffic

- Attack traffic can be generated intentionally or unintentionally.
- In the first case we might prefer to block the source host/network
- In the second case, we might want to block a specific application (malware installed inside our network)
- What is the best policy?
  - A fine-grained policy is more precise, but it requires more memory and CPU resource
  - A coarse-grained policy is easier to manage, but might prevent legitimate services to work
- Impact not only on the mitigation strategy
- But also on the way we process the traffic for intrusion detection

Flow-based processing

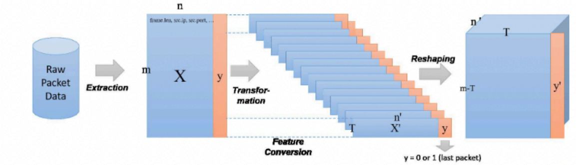
- The most common approach is that of collecting flow-specific features
  - A flow is usually defined by using a tuple like `(srcIP,dstIP,srcPort,dstPort,protocol)`
  - A flow can be represented using packet-based features or statistical features
  - Unidirectional or bi-directional?
    - Bidirectional gives a better view of the interaction between attacker and victim
    - However, it makes it harder to identify the source of the attack

Feature extraction (Packet-level features)

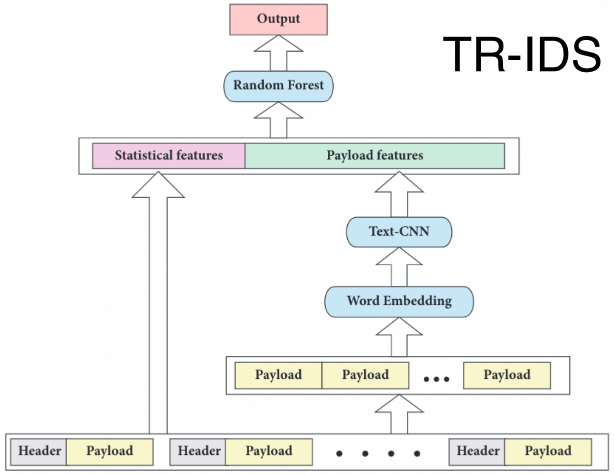
- Packet-level features:
  - Build a representation of a flow grouping representations of packets that belong to the same flow
  - Packet-level info such as header fields (sometimes payload as well)
- IP addresses and ports are usually avoided as features (too specific to the testbed where the data has been collected)

DeepDefense

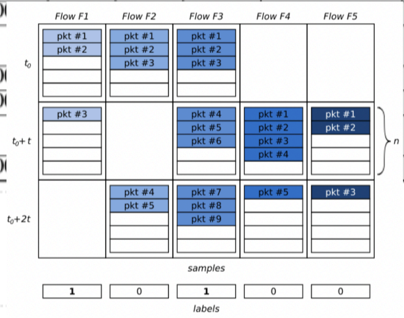
Field	Field Example	Field Type
frame.encap_type	1	Boolean
frame.len	805	Numerical
frame.protocols	eth:ip:tcp:http:data:	Text
tcp.analysis.ack_rtt	0	Numerical
tcp.analysis.bytes_in_flight	1.460e+03	Numerical
tcp.analysis.duplicate_ack_num	1	Numerical
tcp.dstport	2090	Boolean
tcp.flags.urg	0	Boolean
tcp.len	751	Numerical
tcp.srcport	80	Boolean
tcp.window_size	12864	Numerical
udp.dstport	47666	Boolean
udp.length	97	Numerical
udp.srcport	47521	Boolean



TR-IDS



	Pkt #	Time (sec)[1]	Packet Len	Highest Layer[2]	IP Flags	Protocols[3]	TCP Len	TCP Ack	TCP Flags	TCP Window Size	UDP Len	ICMP Type
Packets	0	0	151	99602525	0x4000	0011010001000b	85	336	0x018	1444	0	0
	1	0.092	135	99602525	0x4000	00110100010					0	0
	⋮	⋮	⋮	⋮	⋮	⋮					⋮	⋮
	j	0.513	66	78354535	0x4000	00100100010					0	0
Padding	$j+1$	0	0	0	0	00000000000					0	0
	⋮	⋮	⋮	⋮	⋮	⋮					⋮	⋮
	n	0	0	0	0	00000000000					0	0



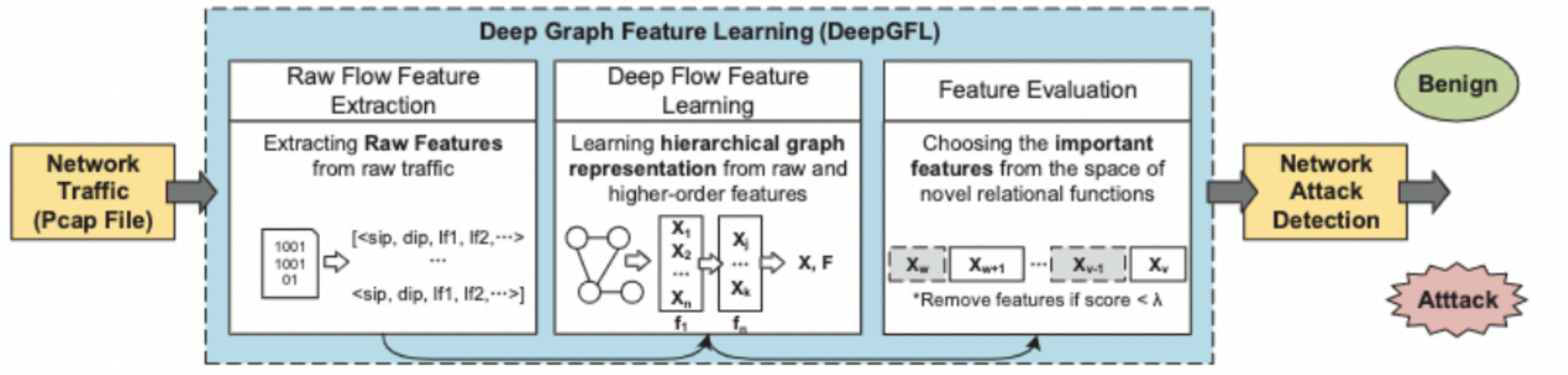
LUCID

Feature extraction (Statistical features)

- Flow-level features:
  - Flows are represented using statistical features
  - Many papers use pre-extracted features provided with the dataset

TR-IDS

Feature	Description
protocol	Protocol of the flow
src_port	Source port
dst_port	Destination port
f(b)_urg_num	Number URG flags in the forward(backward) direction (0 for UDP)
f(b)_ack_num	Number ACK flags in the forward(backward) direction (0 for UDP)
f(b)_psh_num	Number PSH flags in the forward(backward) direction (0 for UDP)
f(b)_rst_num	Number RST flags in the forward(backward) direction (0 for UDP)
f(b)_syn_num	Number SYN flags in the forward(backward) direction (0 for UDP)
f(b)_fin_num	Number FIN flags in the forward(backward) direction (0 for UDP)
pkts_num	Total packets in the flow
bytes_num	Total bytes in the flow
f(b)_pkts_num	Total packets in the forward(backward) direction
f(b)_bytes_num	Total bytes in the forward(backward) direction
f(b)_len_min	Minimum length of packet in the forward(backward) direction
f(b)_len_max	Maximum length of packet in the forward(backward) direction
f(b)_len_mean	Mean length of packet in the forward(backward) direction
f(b)_len_std	Standard deviation length of packet in the forward(backward) direction
duration	Duration of the flow
pkts_psec	Number of packets per second
bytes_psec	Number of packets per second
f(b)_pkts_psec	Number of forward(backward) packets per second
f(b)_bytes_psec	Number of forward(backward) bytes per second
f(b)_intv_min	Minimum time interval between two packets sent in the forward(backward) direction
f(b)_intv_max	Maximum time interval between two packets sent in the forward(backward) direction
f(b)_intv_mean	Mean time interval between two packets sent in the forward(backward) direction
f(b)_intv_std	Standard deviation time interval between two packets sent in the forward(backward) direction



DeepGFL. Feature extraction with CICFlowMeter

Remark

Usability of the traffic representations in real-world deployments

- In real-world application the traffic is collected within a **time-window** and the processed
- Tools like CICFlowMeter detect the beginning and the end of a flow for computing its statistical properties
  - However, we cannot wait the end of an attack to detect it. It would be too late!!!

Length of the time window?

- Depends on the application and on the available resources:
  - Long time-windows allow for **more detailed flow representations**, but require more memory and CPU resources
  - Short time-windows allow for **faster processing and reaction**, but less data is can be collected

Balancing between time-window, amount of incoming traffic and resources is often necessary

- **Dynamic time-windows**, **packet sampling** are techniques that can be used to solve the problem

Feature extraction with tshark and pyshark

- Tshark: https://www.wireshark.org/docs/man-pages/tshark.html
- Tshark lets you **capture** packet data from a live network, or **read** packets from a previously saved capture file, either **printing** a decoded form of those packets to the standard output or **writing** the packets to a file.
- Pyshark: https://github.com/KimiNewt/pyshark
- Python wrapper for tshark, allowing python packet parsing using wireshark dissectors (plugins to decode protocol encapsulations of packets).

Why tshark?

```
protocols = {LayerFieldsContainer} eth:ethertype:ip:tcp
```

- ➢ Easy to debug: its output can be checked in Wireshark
- ➢ High-level packet features: e.g.: list of protocols.
- ➢ Can be used for live-testing (see LUCID code)
- ➢ Supports pcap file format -> compatible with tcpdump

```
Transmission Control Protocol, Src Port: 22607, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 22607
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 1640917568
    [Next sequence number: 1    (relative sequence number)]
  > Acknowledgment number: 800978182
    Acknowledgment number (raw): 800978182
    0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
      > .... .... ..1. = Syn: Set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ··········S·]
    Window size value: 512
```

```
v tcp = {Layer} Layer TCP:\n\tSource Port: 22607\n\tDestination Port: 80\n\tStream index: 0\n\tTCP Segment Len: 0\n\tSequer
     = {LayerFieldsContainer} Layer TCP:\n\tSource Port: 22607\n\tDestination Port: 80\n\tStream index: 0\n\tTCP Segment L
    DATA_LAYER = {str} 'data'
    ack = {LayerFieldsContainer} 800978182
    ack_nonzero = {LayerFieldsContainer} The acknowledgment number field is nonzero while the ACK flag is not set
    ack_raw = {LayerFieldsContainer} 800978182
    checksum = {LayerFieldsContainer} 0x00004e73
    checksum_status = {LayerFieldsContainer} 2
    connection_syn = {LayerFieldsContainer} Connection establish request (SYN): server port 80
    dstport = {LayerFieldsContainer} 80
  > field_names = {list: 37} ['srcport', 'dstport', 'port', 'stream', 'len', 'seq', 'seq_raw', 'nxtseq', 'ack', '_ws_expert', 'ack_non
    flags = {LayerFieldsContainer} 0x00000002
    flags_ack = {LayerFieldsContainer} 0
    flags_cwr = {LayerFieldsContainer} 0
    flags_ecn = {LayerFieldsContainer} 0
    flags_fin = {LayerFieldsContainer} 0
    flags_ns = {LayerFieldsContainer} 0
    flags_push = {LayerFieldsContainer} 0
    flags_res = {LayerFieldsContainer} 0
    flags_reset = {LayerFieldsContainer} 0
    flags_str = {LayerFieldsContainer} \xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7\xc2\xb7S\
    flags_syn = {LayerFieldsContainer} 1
    flags_urg = {LayerFieldsContainer} 0
    hdr_len = {LayerFieldsContainer} 20
    layer_name = {str} 'tcp'
    len = {LayerFieldsContainer} 0
    nxtseq = {LayerFieldsContainer} 1
    port = {LayerFieldsContainer} 22607
    raw_mode = {bool} False
    seq = {LayerFieldsContainer} 0
    seq_raw = {LayerFieldsContainer} 1640917568
    srcport = {LayerFieldsContainer} 22607
    stream = {LayerFieldsContainer} 0
    time_delta = {LayerFieldsContainer} 0.000000000
    time_relative = {LayerFieldsContainer} 0.000000000
    urgent_pointer = {LayerFieldsContainer} 0
    window_size = {LayerFieldsContainer} 512
    window_size_value = {LayerFieldsContainer} 512
```

Feature extraction with Tshark

- Both packet-level and flow-level features can be extracted with tshark/pyshark
- Alternative tools are:
  - **CICFlowMeter** (https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter), a Java-based tool for feature extraction (statistical features).
  - **Tcpdump and libpcap** (https://www.tcpdump.org/), a command line tool and a C/C++ library for network traffic capture and editing (e.g., split).
  - Other tools for pcap file manipulation are
    - **tcprewrite**: packet editing
    - **tcpreplay**: replay a pre-recorded pcap file through a network interface

Feature importance

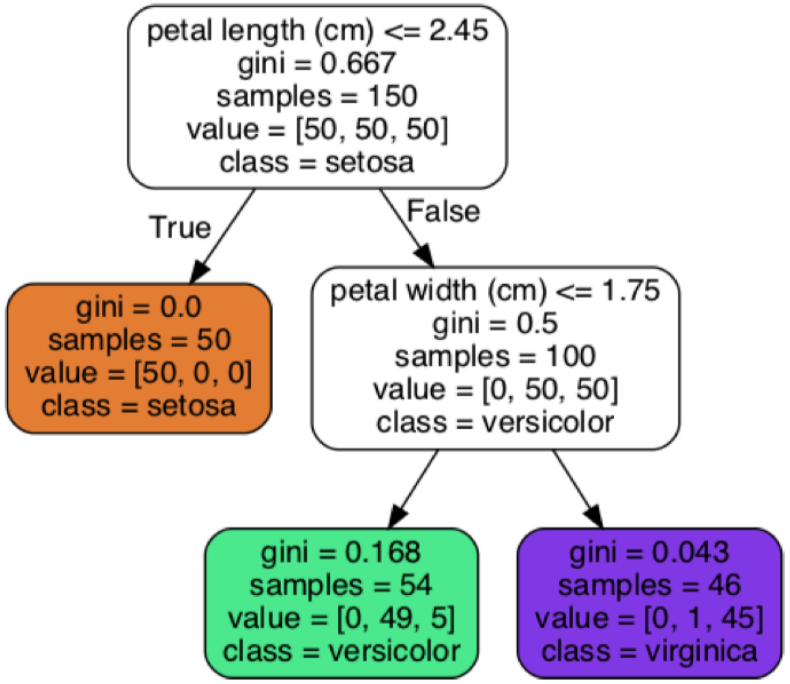
Decision trees(1)

- Decision trees are simple but powerful ML algorithms for classification and regression tasks
- Their decisions are easy to interpret:
  - **samples** counts how many samples a node applies to
  - **value** counts how many training samples of each class the node applies to
  - **gini** is a measure of impurity (gini=0 means pure) and is computed as $G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$

  Where $p_{i,k}$ is the ratio of class *k* samples among the training instances of the i^{th} node.

Example: the Gini impurity of the depth-2 right node is equal to:

$$1 - (0/46)^2 + (1/46)^2 + (45/46)^2 \approx 0.043$$



Iris Decision Tree (source: Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.")

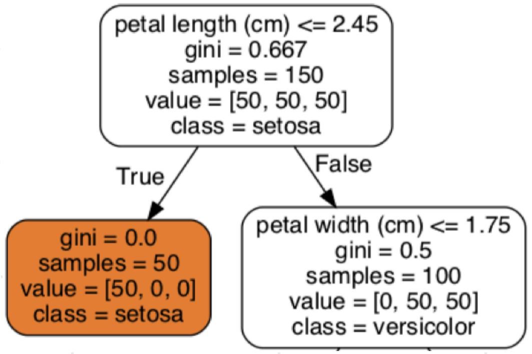
Decision trees (2)

- Scikit-Learn uses the CART algorithm to train DTs
- CART splits the set of each node using a **single feature** k **and a threshold** t_k
- the algorithm chooses **the pair** (k, t_k) **that produces the purest subsets** (lowest gini)

CART cost function for classification: $$J(k, t_k) = \frac{m_{left}}{m} \cdot G_{left} + \frac{m_{right}}{m} \cdot G_{right}$$

Where: $G_{left/right}$ measures the impurity of the left/right subset

$m_{left/right}$ is the number of instances in the left/right subset

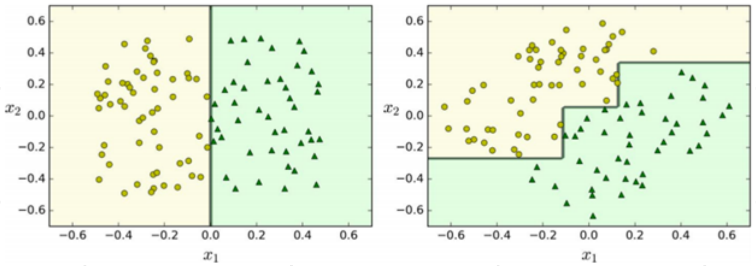


Decision trees (3)

- CART recursively splits each subset until the max depth is reached or the impurity cannot be reduced with further splits
- For regression tasks, CART splits the training set in a way to minimise the MSE
- Decision trees are easy to interpret (often called *white box* models)

Limitations of DTs:

- Sensitive to training set rotation
- Sensitive to small variations of the training set (adding/removing training samples might results in totally different splits)



Source: https://towardsdatascience.com/

Random Forests can limit DT instability by averaging the prediction of many DTs

Random Forests

- A Random Forest is an ensemble of DTs generally trained via **bagging method** (sampling with replacement)
- When splitting a node, RF searches for the best feature among a **random subset of features** (instead of all to reduce the variance)
- Extremely Randomized Trees (Extra-Trees) use **random thresholds** when splitting a node. This reduces the training time (finding the best threshold is time-consuming)
- Classification can be done by assigning the class that obtains the majority of votes (**hard voting**), or assigning the class that obtains the highest average probability (**soft voting**)
- the **scikit-learn implementation** combines classifiers by averaging their probabilistic prediction (**soft voting**), instead of letting each classifier vote for a single class

Feature selection with Random Forest

- Scikit-learn measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity
  - On average across all the tree in the forest
- The values are accessible using the `feature_importances_` variable

Importance feature *i*

Sum over the nodes that use feature *i*

Probability of using a node for a sample

Change in gini impurity at that node

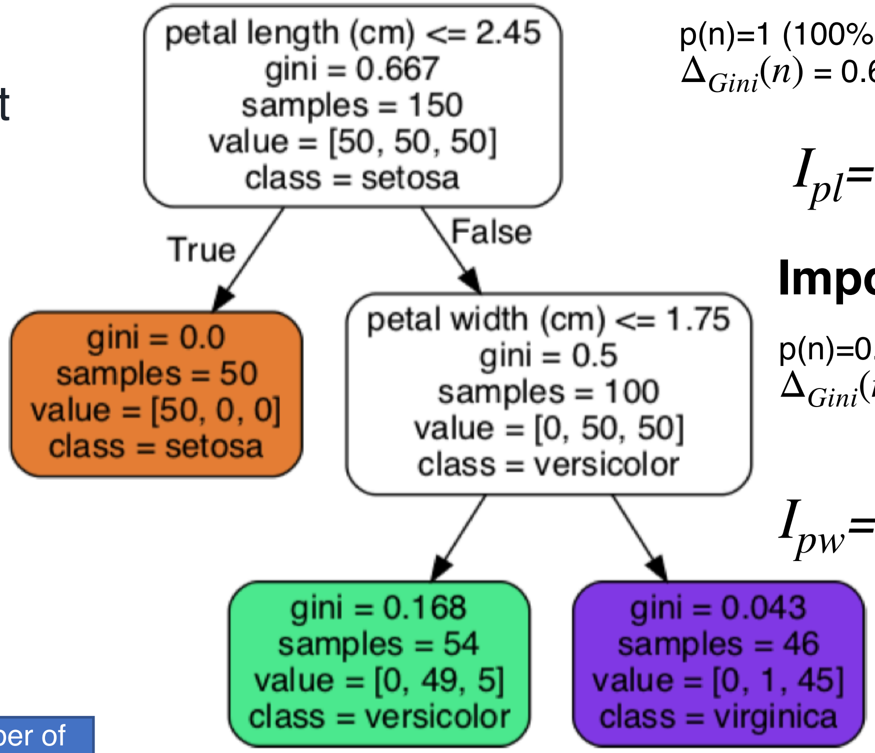
$$I_i(T) = \sum_{n \in T, i_n = i} p(n) \Delta_{Gini}(n)$$

Feature importance with one DT (n=node, i=feature)

$$I_i = \frac{1}{|B|} \sum_{T \in B} I_i(T)$$

Feature importance with RF

Number of trees



Importance of petal length

p(n)=1 (100% of samples)

$\Delta_{Gini}(n)$ = 0.667-(0.334*0+0.667*0.5) = 0.33

I_{pl}=1*0.33=0.33

Importance of petal width

p(n)=0.667

$\Delta_{Gini}(n)$ = 0.5-(0.54*0.168+0.46*0.043)
= (0.09+0.02) = 0.39

I_{pw}=0.667*0.39=0.26

Laboratory: feature selection with RF

- Your problem is to **decide which are the relevant features** for your DDoS detection
- The representation of a flow is in an array-like format, where each row contains a representation of a packet, while each column is a packet-header feature
- A hidden feature is the length of the flow, thus the number of non-zero rows
- This laboratory consists of
  1. transforming the array-like representation of flows into 1-dimensional vectors, with one position representing the length of the flow
  2. Exploring different options for sample's dimensionality reduction (consider the meaning of a feature)
  3. Computing the feature importance with RF

Laboratory: Analisys of model's behaviour

- Once the model is trained, you might want to **understand how your IDS works**
- A basic way to do that is checking which **features are more important** for the classification
- One option is to **set to zero one feature at a time**, a measure the change in model's performance (compare to using all the features)
- If done for all the features, one can understand which ones are more important for the classification
- In the laboratory, you will **re-use the 1D representation** of the flows on the test set to understand which features are more important for a pre-trained model.