Presentation of ShieldNet

by Pascoli Massimiliano
AAU id: 12138922
e-mail: mapascoli@edu.aau.at
date: 12/01/2022

The learning task

ShieldNet aspires to be an effective and efficient tool to detect Denial of Service (DoS) attacks, that would allow a human/machine to take countermeasures manually/automatically with low responsive times. Since the problem is not easy to resolve due to the wide taxonomy of DoS attack techniques, it is beneficial to specify the tuple $\langle Task\_to\_perform, Performance\_measure, Experience\_data \rangle$ that characterize my learning task for the machine learning problem I am going to face with.

⚠ **WARNING**

Please note that every component of the learning task is split in two, this is to account for the **multiple** produced models: there were some technical difficulties with the multi-calss classifcation at first (attempted many different approaches) that got resolved in the end, exploiting the structure and workflow of the binary classification models produced during exploration (this was also a great opportunity to experiment with architectures, hyperparameters, data reduction techniques, ...).

Task to perform

DoS detection via packet flow aggregated metainformation, that is:

- the system must be able to distinguish if the connections flows (traditionally identified by tuple $\langle \mathbf{source\_ip}, \mathbf{source\_port}, \mathbf{destination\_ip}, \mathbf{destination\_port}, \mathbf{protocol} \rangle$) belong to a DoS attack or are just benign traffic. If the flow is malign, possibly know what kind of attack it is (multi-class classification);
    - 12 classes present in the datasets used after the initial clean-up, representing 11 different DoS attacks techniques and the normal traffic class
- there is an easier binary classification variant of the same problem which consists only in classifying between malicious and benign traffic flows starting from the same aggregated data mentioned above.

Performance measure

In general the system should be trained to minimize the number of miss-classifications, both in the binary and multi-class variant. It is particularly important a stong and correct discrimination between benign traffic and all the other malicious classes. If a flow is classified as malign, it should really be, it is less important that the attack technique is recognized correctly (nevertheless a correct classification can be very informative to apply further, more precise, countermeasures).
In the table below are shown the specific metrics (in PyTorch) used to evaluate the goodness of a model after training:

BINARY	MULTI
Per-class Accuracy	Per-class Accuracy
Binary precision	Multiclass precision
Binary recall	Multiclass recall
Binary F1-score	Multiclass F1-score
Binary Overall Accuracy	Multiclass Overall Accuracy

⚠ **WARNING**

Classes will be balanced after the preliminary data analysis. Technically I could have used accuracy as my only meaningful metric, but the other metrics are valid also with unbalanced problems.

Reminder: metrics

		Ground truth	
		0	1
Class prediction	0	True Negative	False Negative
	1	False Positive	True Positive

Metric	**Meaning**
$\text{Precision or Positive Predicted Value (PPV)} = \frac{TP}{TP+FP}$	Out of all *True/Positive/1* predicted values (i.e. class prediction *True/Positive/1*), how many of them were really *True/Positive/1* in %?
$\text{Recall or True Positive Rate (TPR), Sensitivity, Hit Rate} = \frac{TP}{TP+FN}$	Out of all samples that were really *True/Positive/1* (i.e. ground truth *True/Positive/1*), how many were predicted as *True/Positive/1* in %?
$\text{F1-score} = \frac{2}{\frac{1}{TPR}+\frac{1}{PPV}}$	Harmonic mean of Precision and Recall
$\text{Accuracy (ACC)} = \frac{TP+TN}{TP+TN+FP+FN}$	Out of all samples, how many were correctly classified in %?
Per-class Accuracy	Correctly predicted for class i (i.e. prediction equals ground truth) over total samples with same ground truth class i

Experience data

In order to train the models, two different datasets were used:

- for the multi-class classification problem **CIC-DDoS2019** was used;
- for the binary classification problem **this** dataset was used (ddos_balanced.csv only), which is a balanced mixture of data coming from other very popular IDS datasets, in particular
    - **CSE-CIC-IDS2018-AWS**;
    - **CICIDS2017**;
    - **CIC DoS dataset(2016)**.

Both datasets contain information of many generated internet traffic flows, labeled with *benign*, a specific attack name or general *DoS*. The data I used was generated with the same learning task in mind: it was recorded on a test network diverse enough to include several possible attack devices and victim devices. The data was recorded capturing raw packets using **libpcap** (OS independent library that is available for use in many programming languages) available with *tcpdump* utility and then, using **CICFlowMeter** some aggregated metrics were computed (please refer to **this page** to know what metrics this tool can calculate given raw packets in pcap standard). Around 80 of these metrics were chosen to be present in the datasets and not all of them are useful for my goal, nor for every model proposed in the Implementation section.

⚠ **WARNING**

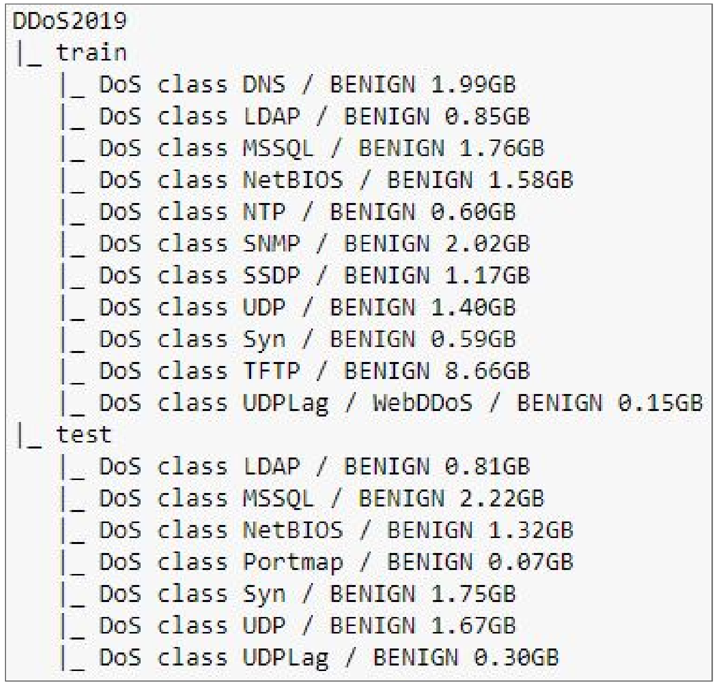
Datasets contain **flow data**: around 80 metrics for every flow were computed from many (millions, bllions, ...) internet packets belonging to the same flow! Technically, a single flow is directional ($flow(A \to B) \neq flow(B \to A)$), but in both datasets are considered bidirectional!

Examining the datasets

In this section, my goal is to understand the data, its organization and representation, in order to get ready for learning.
In the end we would like to have more or less **balanced** datasets, this is to help learning a correct inference model/function, for both the multi-class classification task (MCCT) and binary classification task (BCT).
Furthermore, it is important for the purposes of training that the data we provide to the ANN is **not malformed or missing** and is **numeric**.

MCCT dataset analysis

The schematic structure of CIC-DDoS2019 dataset is the following:

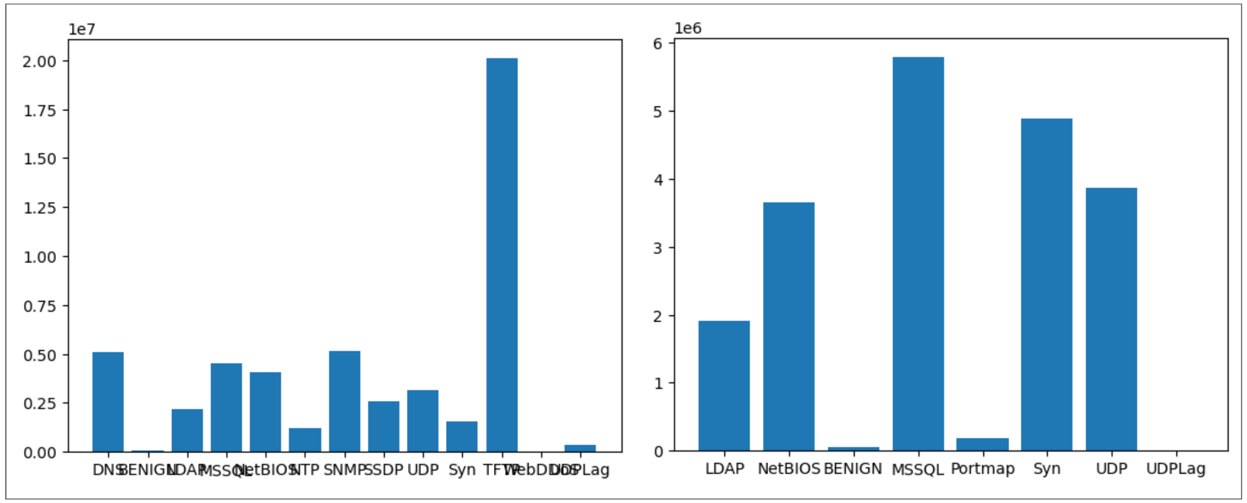


```
DDoS2019
|_ train
   |_ DoS class DNS / BENIGN 1.99GB
   |_ DoS class LDAP / BENIGN 0.85GB
   |_ DoS class MSSQL / BENIGN 1.76GB
   |_ DoS class NetBIOS / BENIGN 1.58GB
   |_ DoS class NTP / BENIGN 0.60GB
   |_ DoS class SNMP / BENIGN 2.02GB
   |_ DoS class SSDP / BENIGN 1.17GB
   |_ DoS class UDP / BENIGN 1.40GB
   |_ DoS class Syn / BENIGN 0.59GB
   |_ DoS class TFTP / BENIGN 8.66GB
   |_ DoS class UDPLag / WebDDoS / BENIGN 0.15GB
|_ test
   |_ DoS class LDAP / BENIGN 0.81GB
   |_ DoS class MSSQL / BENIGN 2.22GB
   |_ DoS class NetBIOS / BENIGN 1.32GB
   |_ DoS class Portmap / BENIGN 0.07GB
   |_ DoS class Syn / BENIGN 1.75GB
   |_ DoS class UDP / BENIGN 1.67GB
   |_ DoS class UDPLag / BENIGN 0.30GB
```

As you can see, the raw dataset is difficult to use for at least 4 reasons:

1. BENIGN class is not separated in its own file, but it is mixed with all the other classes in very csv;
2. Not all classes present in the training part are present also in the test part (and vice-versa): the training/test split suggestion is useless for our task, might as well ignore it completely;
3. Classes are far from balanced;
4. The size of every csv listed ranges from 0.1GB to 8.7GB given a total of 28.9GB: impossible to load all dataset at one or only training/test part to have a more or less complete view.

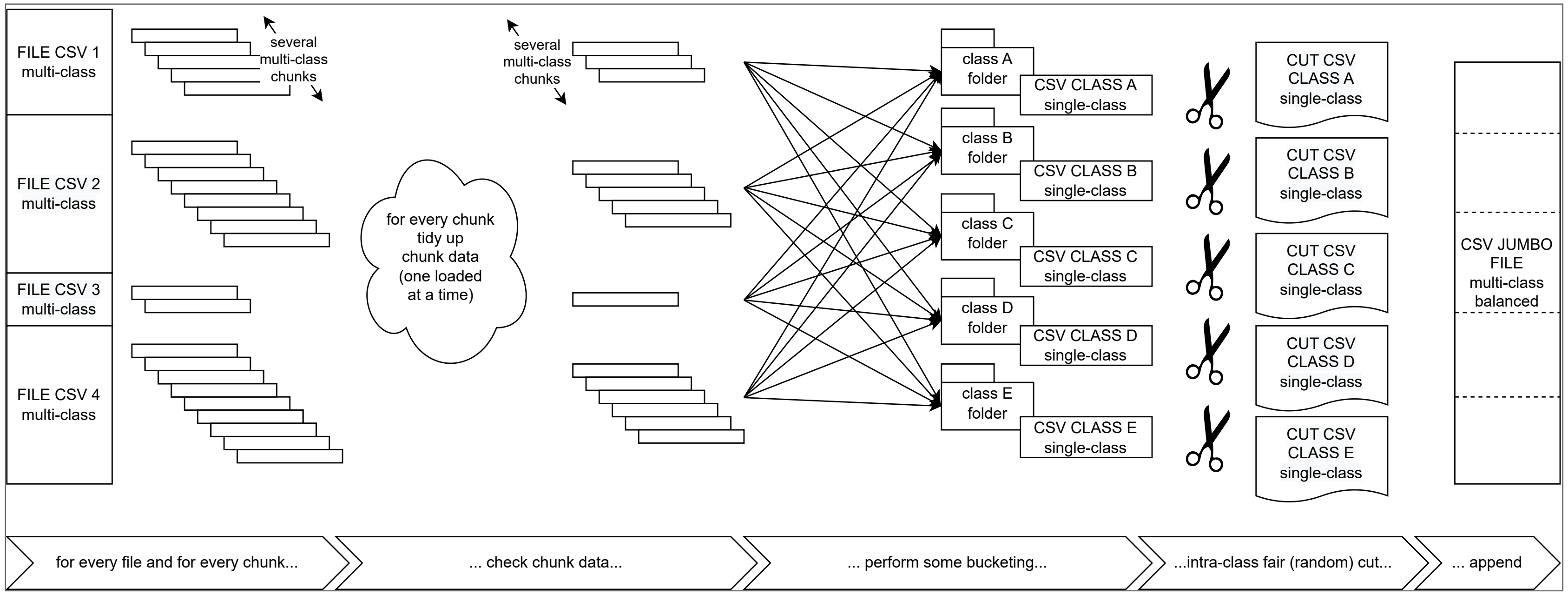
Classes distribution of CIC-DDoS2019 in detail for point 3 presented previously (training part left, test part right):



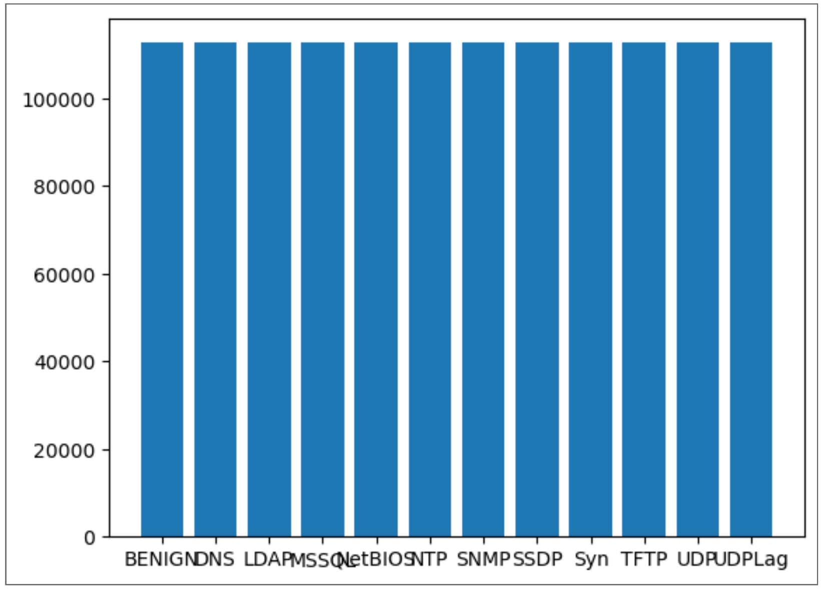
The number of samples for each class is completely out of scale with respect to others.
WebDDoS and Portmap traffic samples are almost non-existant compared to BENIGN class, and Portscan is technically not a DoS technique, but a service and port probing tool...
WHAT DO WE DO?

Solution to MCCT dataset

Given the dataset in the form described and since my computing power is limited, the solution implemented is to reconstruct a single, unified, balanced dataset cutting down in a **fair** manner all oversized classes, dropping Portmap and WebDDoS classes entirely. Below, a brief data transformation schema is reported:

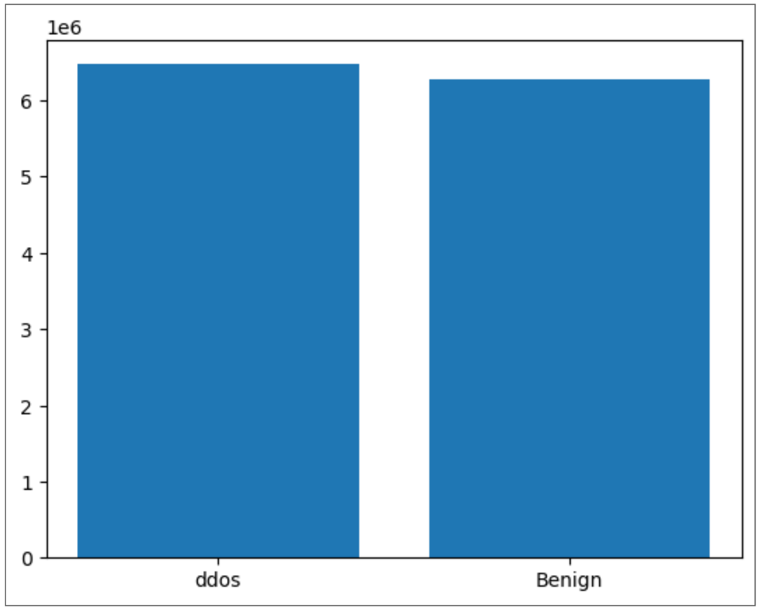

FILE CSV 1 multi-class
FILE CSV 2 multi-class
FILE CSV 3 multi-class
FILE CSV 4 multi-class
several multi-class chunks
for every chunk tidy up chunk data (one loaded at a time)
several multi-class chunks
class A folder
CSV CLASS A single-class
class B folder
CSV CLASS B single-class
class C folder
CSV CLASS C single-class
class D folder
CSV CLASS D single-class
class E folder
CSV CLASS E single-class
CUT CSV CLASS A single-class
CUT CSV CLASS B single-class
CUT CSV CLASS C single-class
CUT CSV CLASS D single-class
CUT CSV CLASS E single-class
CSV JUMBO FILE multi-class balanced
for every file and for every chunk...
... check chunk data...
... perform some bucketing...
...intra-class fair (random) cut...
... append

The final result is a csv file in which samples belonging to every class are cut down to the number of samples that the least numerous class has.
As a collateral product, not only the dataset has the desiderable properties described, but also is only 0.43GB. This allows me to load all data to main memory at once as a single DataFrame (empirically main memory occupied by DataFrame object is two times the csv file size).
Every class now has 112731 samples, bringing the total number of samples to $112731 \cdot 12 = 1352772$ as shown below:

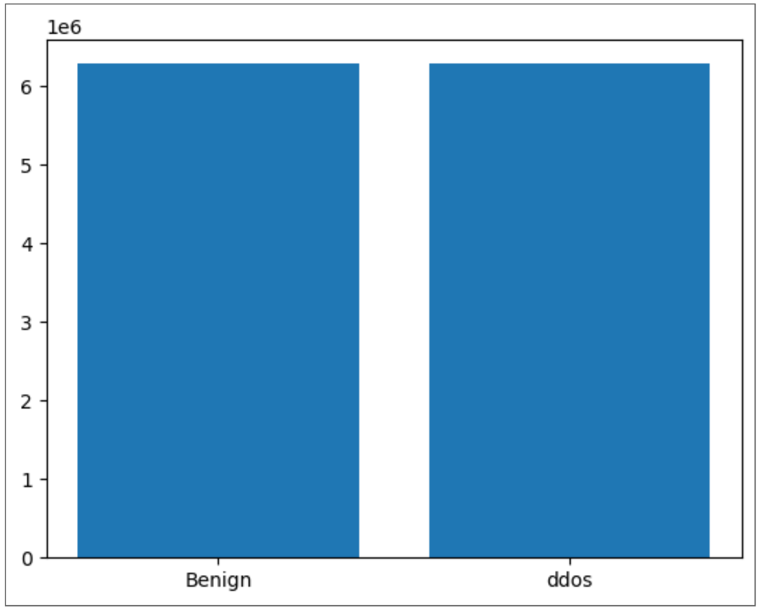


Solution to BCT dataset

Here the problem is slightly different, but the solution and the code used is exactly the same. This dataset is composed by one file only that has all the labeled flows samples I am going to use". This file contains almost the same columns as the MCCT dataset (in the result there will be the same columns exept for different naming conventions...).
It also starts pretty balanced as you can see, but its size is prohibitive (6.32GB):



After the application of my fair cut utility:



Unfortunately the resulting size is huge nontheless: 5.12GB for $6274230 \cdot 2 = 12548460$ samples. For initial exploratory tries in PyTorch I will work on a subset of this result csv (around 1000000 samples).

Literature overview

Implementation

Results

```
In [7]: torch.cuda.is_available()
```

```
Out[7]:

True
```

```
In [ ]:
```