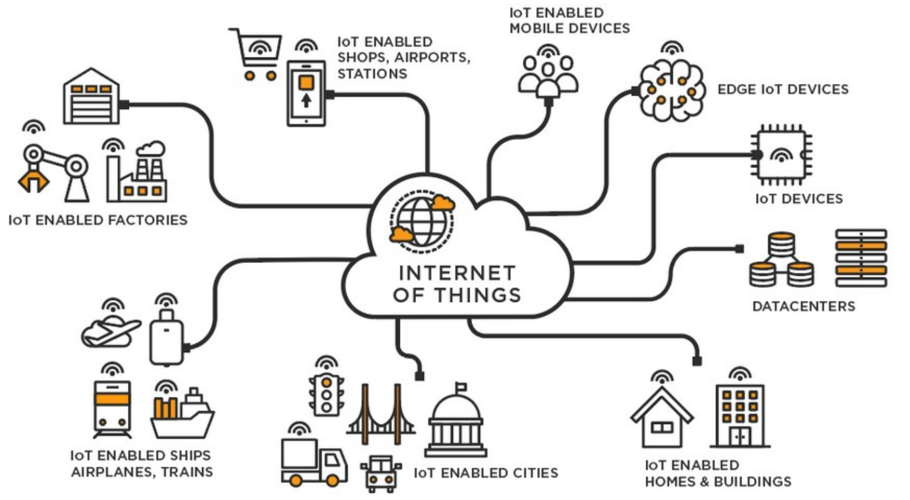
Anomaly detection on network traffic from Internet of Things (IoT) devices

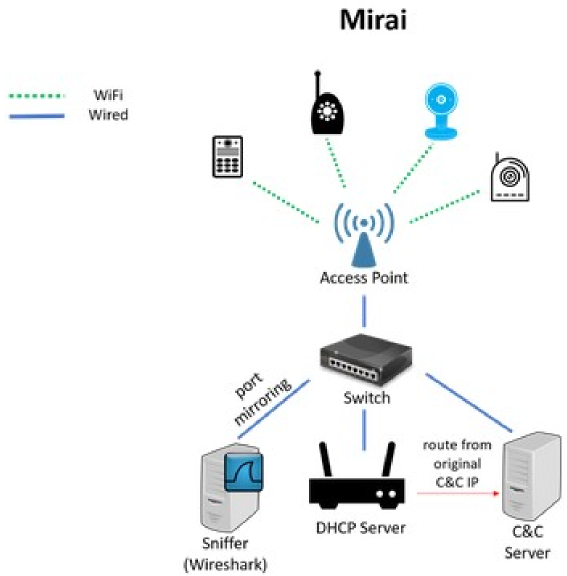
Ourania Sidiropoulou

Capstone project defense

21/07/2022

Introduction





Facts:

- provide great level of automation and information sharing to enhance usability and functionality
- rapid growth the last years

The problem:

- number of threats and cyber-attacks is constantly growing
- concerns related to network security and data privacy as most of these networks are not usually well protected.

Project Goal:

Can we distinguish with high confidence the different type of flows, benign or malicious, in network activities?

Data Availability & Processing

A publicly available **labeled dataset**[1] of network traffic from IoT devices with:

- 3 benign captures → *all processed*
- 20 malware captures → *processed: 7/20*

merged into a single csv file

for binary and/or multi-class classification scenarios → *focused only on* ***multi-class***

Malware captures

Botnet	Malware_type	counts
Hakai	C&C	8222
	benign	2181
Mirai	C&C	6720
	C&C-FileDownload	11
	DDoS	14395
	PartOfAHorizontalPortScan	122
	benign	2134
Muhstik	Attack	5962
	C&C	8
	PartOfAHorizontalPortScan	145597
	benign	4536
Torii	C&C-Torii	30
	benign	6465
Trojan	C&C-FileDownload	3
	FileDownload	3
	benign	4420

contain also benign entries

IoT devices

1) home intelligent personal assistant

2) smart LED lamp

3) smart door lock

(1) Stratosphere Research Laboratory, CTU University, Czech Republic

EDA: Data Overview

- # **entries**: **202,765**
- # initial features: 23 but ...
  - **3** → NaNs
  - **1** → 94% NaNs
  - **2** → not useful for ML task (uid & timestamp)
- # **features**: **17**
  - 7 categorical — 5 input / 2 targets (binary/multi-class classification)
  - 10 numerical
- **Missing values in 7 features:**
  - easy to be replaced, will not be discussed (in back up slides)

#	Column	Non-Null Count	Dtype	
0	timestamp	202765 non-null	datetime64[ns]	not useful
1	uid	202765 non-null	object	not useful
2	origin_address	202735 non-null	object	
3	origin_port	202765 non-null	int64	
4	response_address	202765 non-null	object	
5	response_port	202765 non-null	int64	
6	protocol	202765 non-null	object	
7	service	13235 non-null	object	~94% NaNs
8	duration	101394 non-null	object	wrong encoding as object type
9	orig_bytes	101394 non-null	object	
10	resp_bytes	101394 non-null	object	
11	conn_state	202765 non-null	object	
12	local_orig	0 non-null	float64	All empty values
13	local_resp	0 non-null	float64	
14	missed_bytes	202765 non-null	int64	
15	history	201422 non-null	object	
16	orig_pkts	202765 non-null	int64	
17	orig_ip_bytes	202765 non-null	int64	
18	resp_pkts	202765 non-null	int64	
19	resp_ip_bytes	202765 non-null	int64	
20	tunnel_parents	166506 non-null	object	Non-null = (empty)
21	label	202765 non-null	object	
22	detailed_label	181073 non-null	object	

	count	unique	top	freq
tunnel_parents	166506	1	(empty)	166506

EDA: Features and target values

f: input features
t: target values

<u>Data exchange through a **protocol (f5)**:</u>

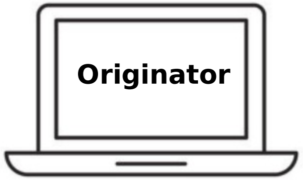
bytes (**f6**, **f7**)
IP bytes (**f8**, **f9**)
packets (**f10**, **f11**)

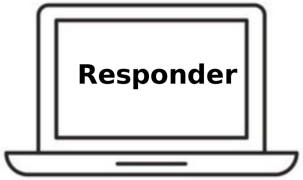
Correlated features

missing bytes (**f12**)

<u>Other info captured by monitoring system:</u>

connection state (**f13**)
history of connection (**f14**)
duration of connection (**f15**)





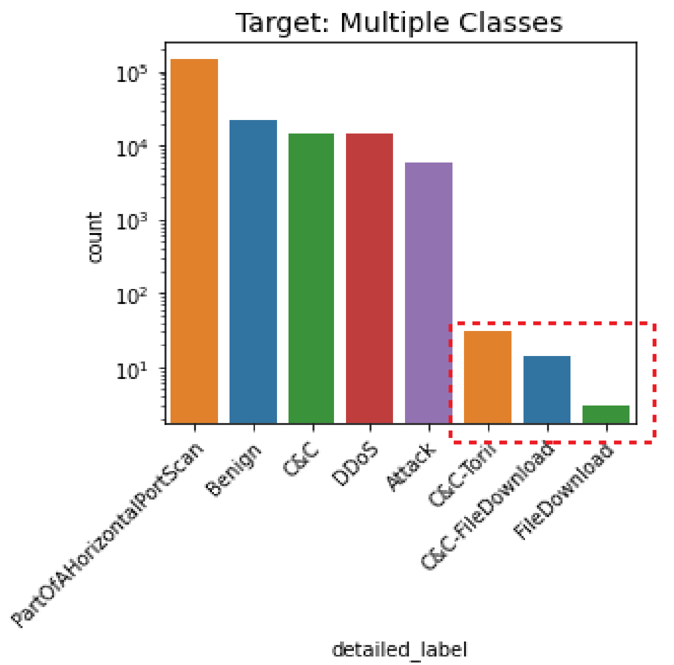
IP address (IPv4 or IPv6):port number
f1:**f2**

IP address (IPv4 or IPv6):port number
f3:**f4**

<u>Type of traffic:</u>

t1: benign or malicious **(label)**
t2: type of malware when malicious **(detailed label)**

EDA: Type of Classes



Classes are highly imbalanced:

- Horizontal port scans: more than half of total entries
- C&C-Torii
- C&C-FileDownload
- FileDownload

47 entries in total → **dropped** as they will cause ill-conditioned input matrices when splitting the dataset

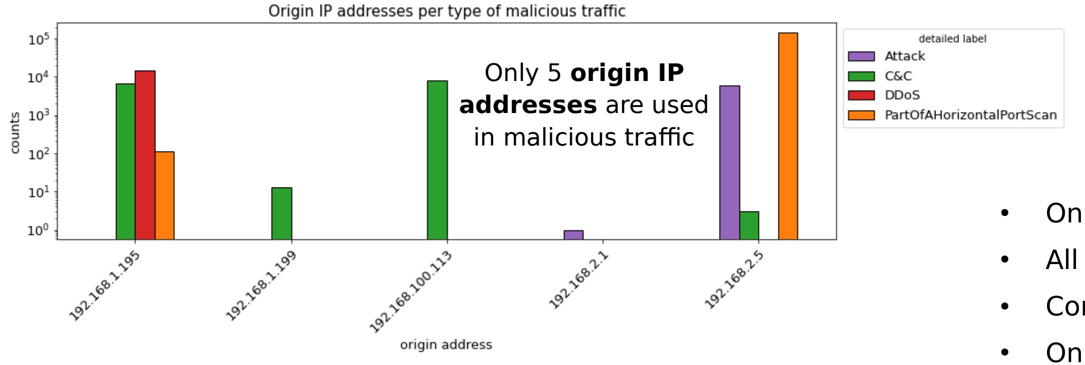
Type of malware:

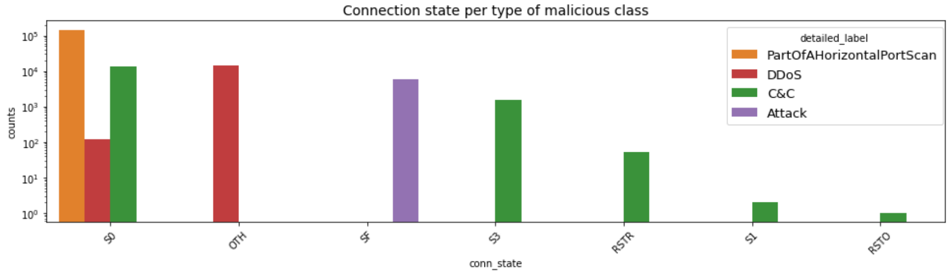
1) *Horizontal port scans (PartOfAHorizontalPortScan)*
2) *Command and Control (C&C)*
3) *Distributed Denial-of-Service (DDoS)*
4) *FileDownload (either connected through a CC server or not)*
5) *Torii (characteristic of Torii botnet)*
6) *Attack (some type of attack from infected device to another host)*

EDA: Categorical Features

Summary Table: #uniques per categorical feature

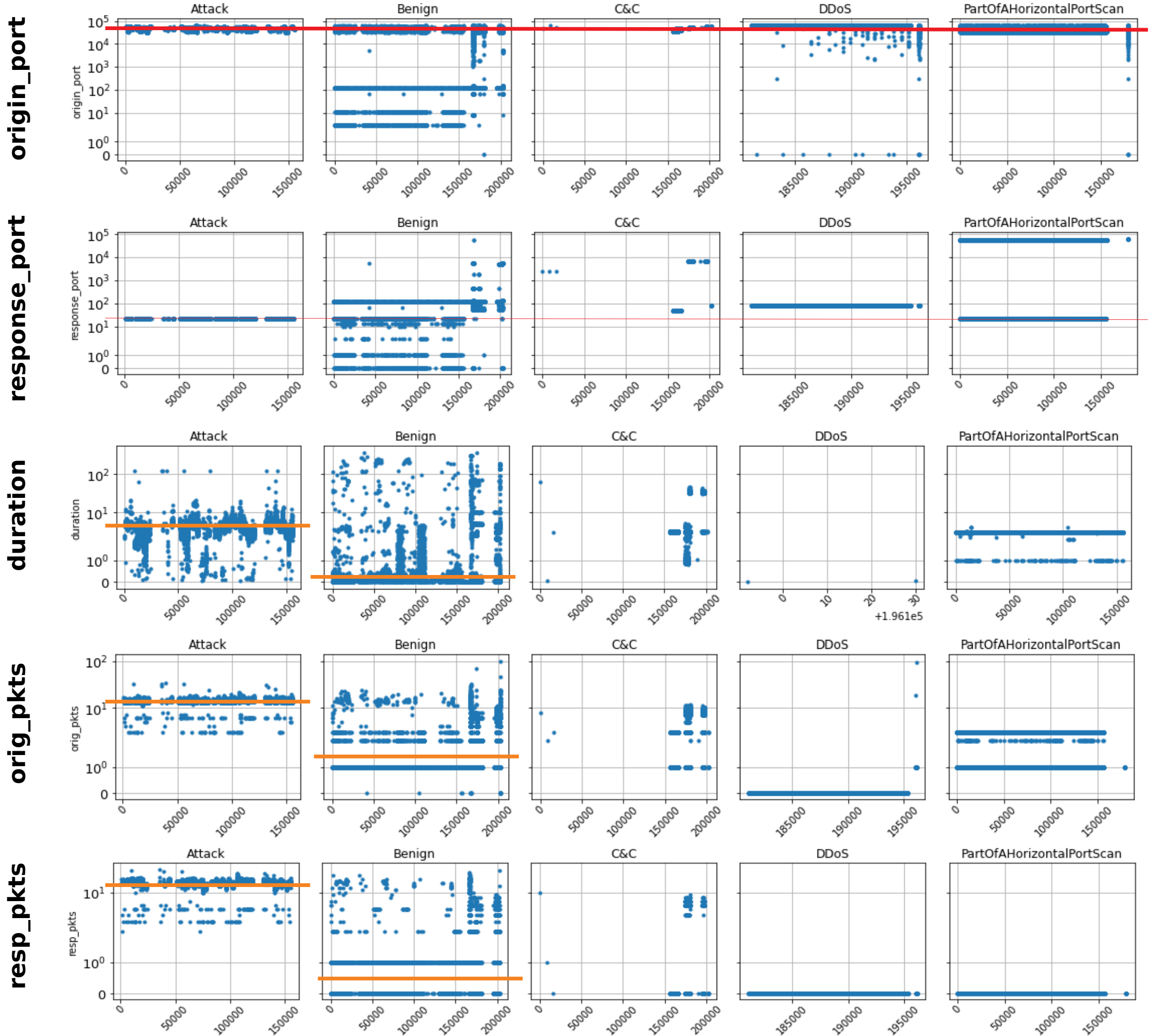
Class	#unique IP's Originator (total=1000)	#unique IP's Responder (total = 64,420)	Protocol (#uniques = 3)	#unique History (total = 105)	#unique connection state (total = 12)
Benign	**1,000**	~3,000	**TCP**, UDP or ICMP	65	**12**
HPortScan	2	**64,420**	**TCP**	1	1
C&C	4	8	**TCP**	11	5
Attack	2	**200**	**TCP**	30	1
DDoS	1	2	**TCP**	4	2





- Only 5/1000 origin IP's are used for malicious traffic
- All response IP's are used by H.PortScan
- Common protocol for all classes is the TCP
- Only 6/105 type of histories used for both benign & malicious traffic
- Only 1 type of connection state is common for H.PortScan, DDoS and C&C

EDA: Numerical Features

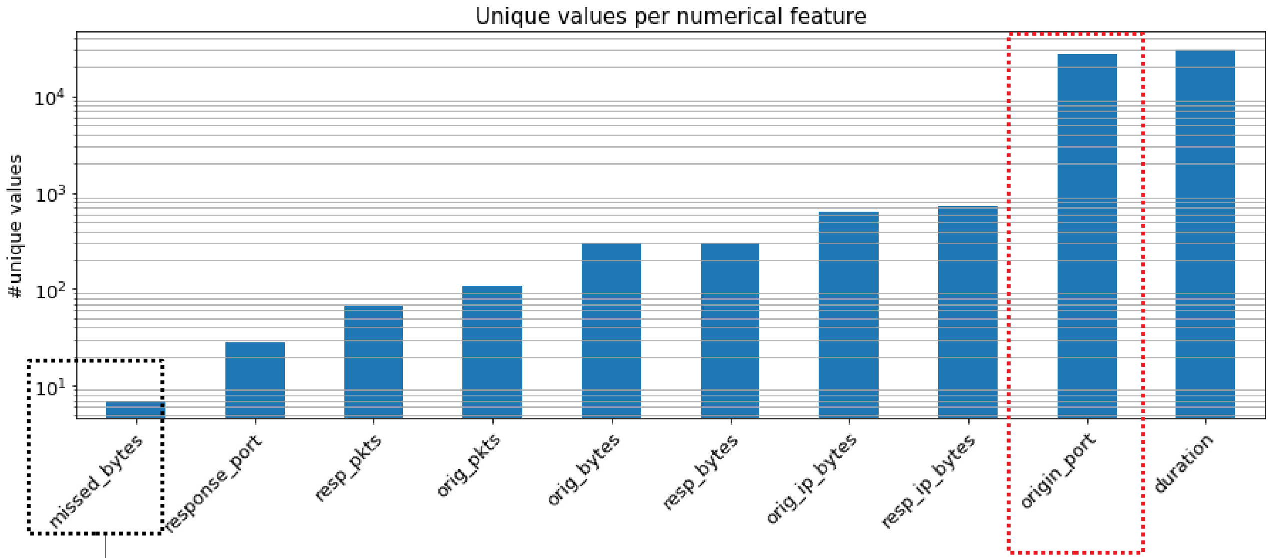


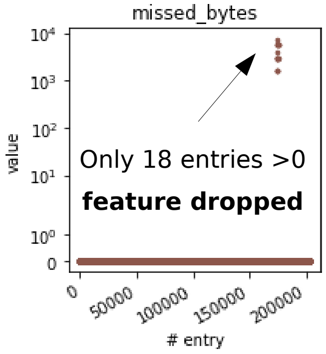
- There is a common range of **origin ports** used by all classes except the DDoS (mainly 65,000 not used by others) but the **response ports** are quite unique.
- A classifier could be confused among Attack, Benign and H.Port scans wrt the response ports alone but:
  - the **duration** of connection is unique for the H.Port Scans.
  - in between Attack and Benign though the median of duration differs, as well as the **packets** send by both originator and responder.

no derived features are necessarily needed to ease the classifiers as we will also see later

— *represents common values*
— *median*

EDA: Numerical Features

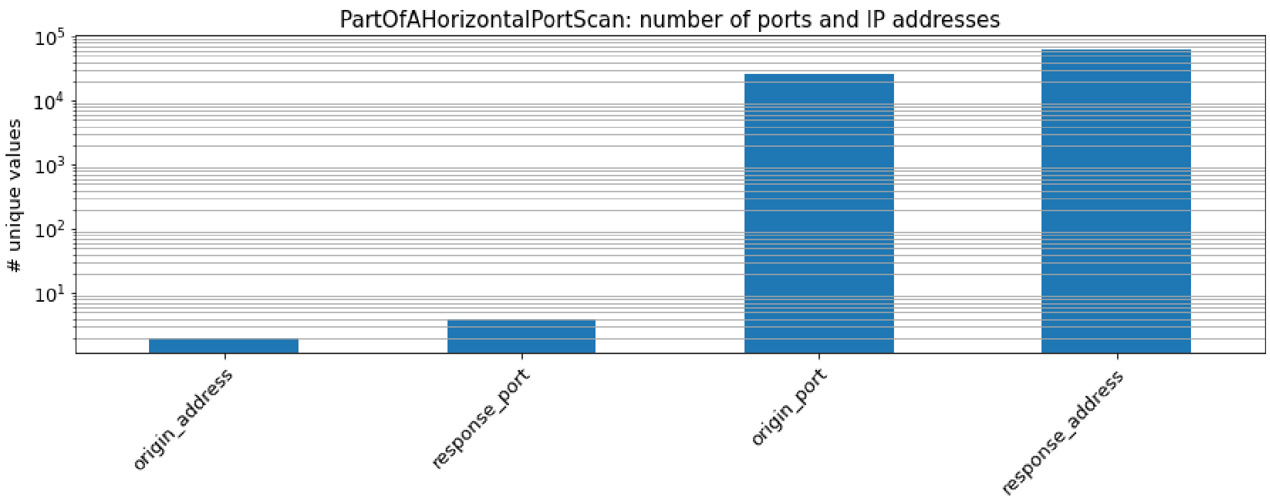




need to be checked if most of these ports are used by the Horizontal Port Scan, as

- **Horizontal port scan:** single port → many IP's
- **Vertical port scan:** single IP → many ports

EDA: PartofAHorizontalPortScan Investigation



- From the above is clear that we have both type of port scans
- To be able to check the scans, we can get the **combination of** the features **origin_address** and **response_port** which have only a few entries:

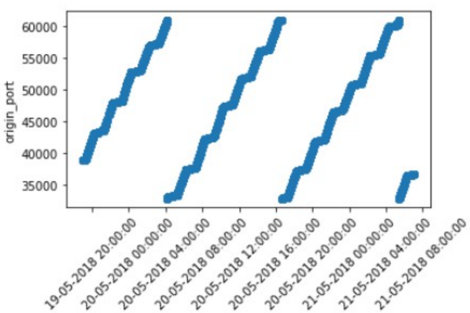
```
# Display frequencies from origin IP addresses and response ports
df = data_df.loc[data_df['detailed_label']=='PartOfAHorizontalPortScan']
df[['origin_address', 'response_port']].value_counts()
```

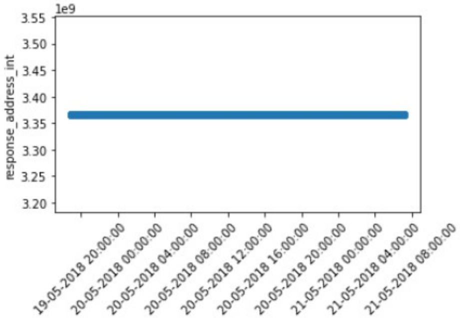
```
origin_address  response_port
192.168.2.5     22               122302
                59353             23295
192.168.1.195   63798               111
```

3 combinations

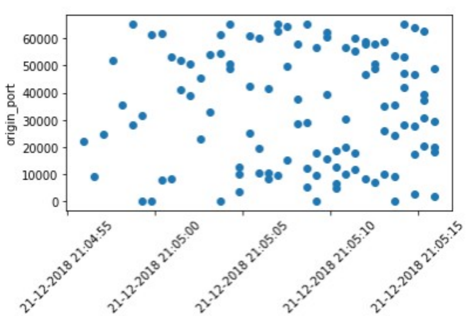
EDA: PartofAHorizontalPortScan Inspection

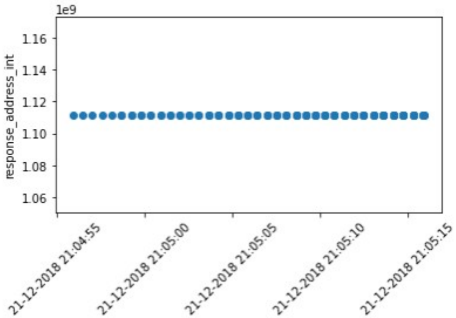
```
origin_address: 192.168.2.5 , response_port: 59353

Number of entries: 23295

Number of unique values:
origin_port          10317
response_address         1
dtype: int64
```






```
origin_address: 192.168.1.195 , response_port: 63798

Number of entries: 111

Number of unique values:
origin_port          105
response_address       1
dtype: int64
```





- **Horizontal**: single port → many IP's
- **Vertical**: single IP → many ports

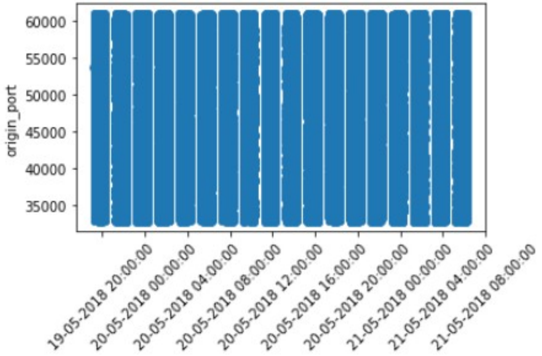
In 2 out of the 3 combinations:

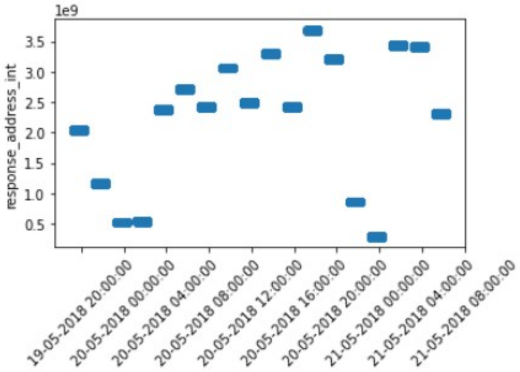
- single response IP → many ports = **Vertical Scans!**

Defined a correction with these combinations of origin addresses and response ports called "1st correction"

EDA: PartofAHorizontalPortScan Inspection

```
origin address: 192.168.2.5 , response port: 22

Number of entries: 122302

Number of unique values:
origin_port          25093
response_address     61765
dtype: int64
```



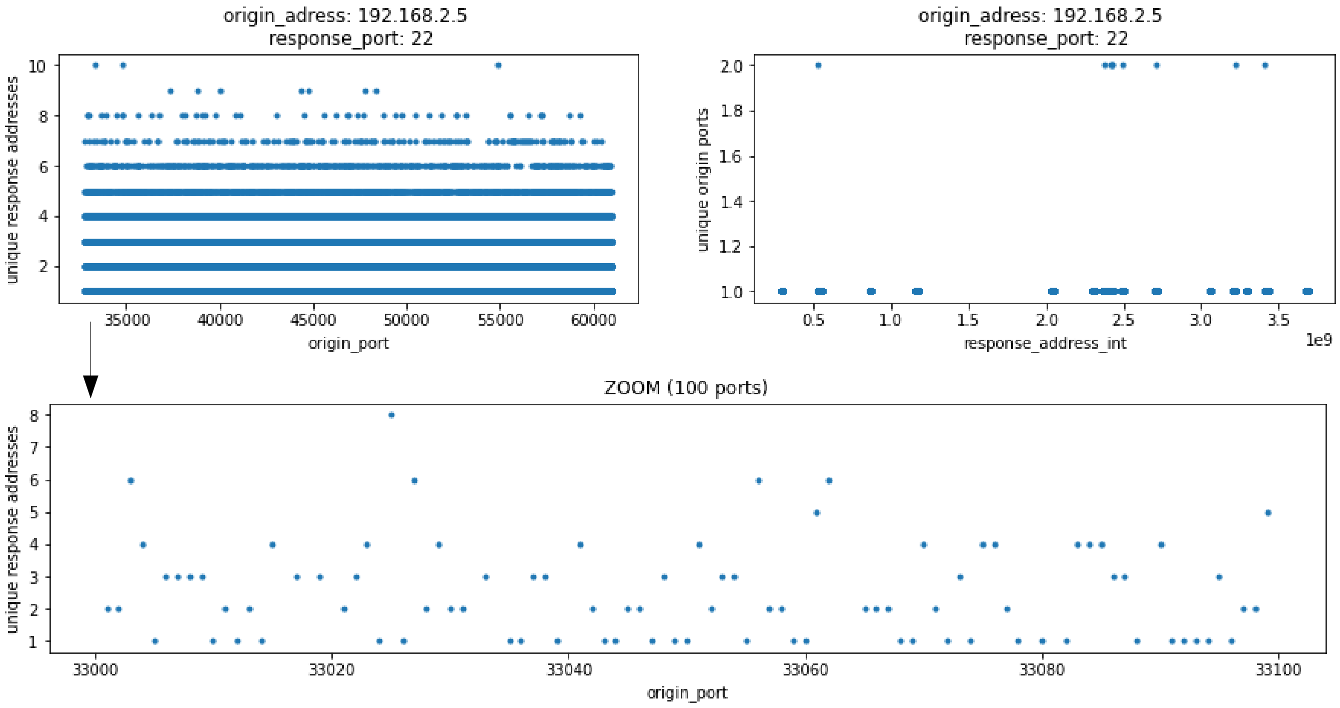


- **Horizontal**: single port → many IP’s
- **Vertical**: single IP → many ports

For the 3rd combination there are 11 distinct captures:

- for each one it seems that there is a single response IP address and many different ports which would imply **vertical scans**
- BUT #unique response_address = 61,765 ≠ 11 as it seems at a first glance
- either horizontal scans OR a mix of horizontal and vertical

EDA: PartofAHorizontalPortScan Inspection



- **Horizontal**: single port → many IP’s
- **Vertical**: single IP → many ports

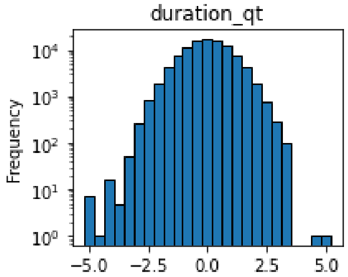
From the above we see that:

- for each origin port there might be 1 or 2 or ... or 10 unique response addresses and
- for each response address there might be 1 or 2 different origin ports

One could assume when:

- single origin port → #unique response address > 1 → **Horizontal Scans**
- #unique response address = 1 → **Vertical Scans** (“2nd correction”)

Feature Engineering



- **Numerical features →** all except ports, pick close to zero and are skewed towards their max values
  - **Log, power and quantile transformations →** none of them distributes features in a better way except the quantile transformation of the feature duration which is normally distributed.

- **Binary features from ip address lib** as for e.g. if an IP address is:

x2

1) private
2) global
3) unspecified → False for responder
4) reserved
5) loopback → False for both originator and responder
6) multicast → False for originator

- 4/12 have only False values → not useful
- the others might be correlated with the target

- **Encoding IP addresses into octets**
  - According to E.Shao[1], the best way of encoding IP addresses for network intrusion detection is to split them in pairs of bits
    - e.g. 192.168.1.1 → x_1=192, x_2=168, x_3=1, x_4=1
  - IPv6 addresses which are in 211 total and used only for benign traffic were dropped since they will create 4 additional features with 0 values for all the IPv4

(1) *Shao, Enchun (2019): Encoding IP Address as a Feature for Network Intrusion Detection. Purdue University Graduate School https://doi.org/10.25394/PGS.11307287.v1*

Correlation of Features with Target

The linear correlation among all features has been checked with the pearson method after encoded rest of categorical features (connection state, history & protocol) as enumerated.

Correlations with the target:

- **High** (0.7 - 0.9): 4 features → **2** from added features
- **Medium** (0.3 - 0.5): 7 features → **2** from added features
- **Low** (-0.2 – 0.2): 11 features → **6** from added features

No.	Feature	Correlation with target	
1	**orig_address_oct4**	0.9	high
2	orig_pkts	0.8	
3	history	0.7	
4	**resp_address_oct1**	0.7	
5	origin_port	0.5	medium
6	response_port	0.5	
7	origin_address	0.4	
8	**orig_address_oct3**	0.4	
9	**resp_address_oct4**	0.4	
10	response_adress	0.3	
11	protocol	0.3	
12	orig_ip_bytes	0.2	low
13	resp_pkts	0.2	
14	**origin_is_private**	-0.1	low
15	**origin_is_global**	-0.1	
16	duration	-0.2	
17	orig_bytes	-0.2	
18	conn_state	-0.2	
19	**resp_is_private**	-0.2	
20	**resp_is_global**	-0.2	
21	**resp_address_oct2**	-0.2	
22	**resp_address_oct3**	-0.2	

Color scale: 1 … -1

Next Step

Tests performed with a classifier in order to understand:

1) what is the best encoding of the IP addresses, integers or octets? → integer encoding is performed entry by entry with the ip address library and it takes quite some time, slowing down the pre-processing
2) rest of categorical features encoded as enumerated or dummies? → $PCC_{history}$ = 0.7 with history as enumerated
3) if the transformed numerical features and binary features improve the classification task → not really expected to see an improvement as PCC where quite low.
4) if disentangling vertical from horizontal port scans does not confuse a classifier and improves the classification task

For these tests, a non-linear classifier was used to be able to generalize them and take a decision for the final pre-processing steps.

- **Random Forest** → a bagging ensemble method that tries to reduce the variance of several decision tree (DT) classifiers and uses averaging to improve the predictive accuracy and control over-fitting of DT's:
  - **weights** → associated with each class can be adjusted to tackle the imbalance of the classes.
  - **number of trees** → set to 10
  - **max_features** → set to None in order for all the features to be taken into account
  - **bootstrap flag** → turned off in order for the whole dataset to be taken into account → DT's are always the same in this case
  - **random_state** → set to 0

Dataset splitting with ***train_test_split*** and ***stratify = y*** to preserve the percentage of samples for each class:

- initially 70-30 for training and testing (random state=0) → always like that
- training set split into 4 different pairs (80-20, 70-30, 60-40, 50-50) for training and validation over 10 different random states

Test sample is not used for these tests

Pre-processing steps

- Pre-processing steps have been implemented with custom transformers, inheriting from BaseEstimator and TransformerMixin classes
- Pipeline defined once, parameters of steps were changed accordingly to the test

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier

# Categorical Features for transformation
cols_to_dummies = ['protocol', 'conn_state', 'history']

# Define the pipeline
pipe = Pipeline([('recover_nulls', pre.RecoverNansPreprocessor()),
                 ('cleaning_preprocessor', pre.CleaningPreprocessor()),
                 ('categorical_preprocessor', pre.CategoricalPreprocessor(cols_to_dummies = cols_to_dummies,
                                                                          cols_to_numeric = [])),
                 ('numerical_preprocessor', pre.NumericalPreprocessor(cols_to_logs = [],
                                                                      cols_to_pt = [],
                                                                      cols_to_quantile = [],
                                                                      replace = False)),
                 ('add_binaries', pre.AddBinariesPreprocessor(has_ip_address_features=False)),
                 ('ip_encoding', pre.IPEncodingPreprocessor(ip_to_octets = True)),
                 ('clf', RandomForestClassifier(n_estimators=10, class_weight='balanced_subsample',
                                                max_features=None, bootstrap=False, random_state=0))
])
```

Replace their ‘nulls’ [-, :] with np.nans

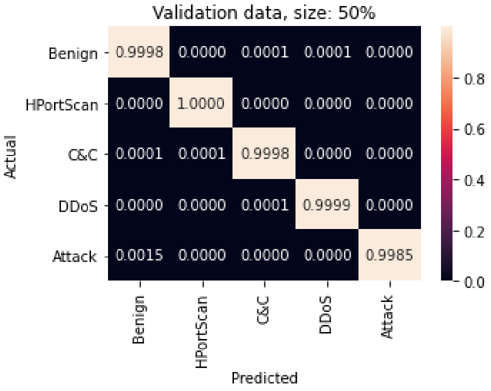
Drop unnecessary columns and replace nulls

Steps for testing

Tests over EDA & Feature Engineering

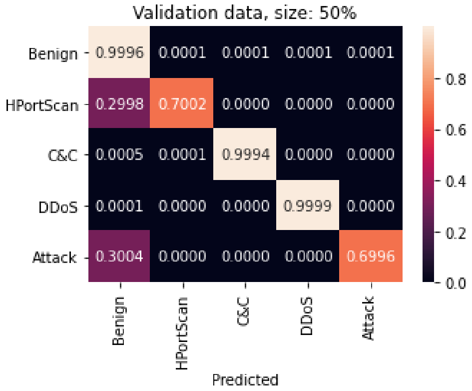
No.	Test	Results
1	Encoding of IP addresses as octets or integers ($PCC_{(some\ octets,\ target)}$ = 0.4--0.9)?	same
2	Categorical encoded as enumerated?	high miss-classification of HportScan and Attack as Benign
3	Only history as enumerated ($PCC_{(history,\ target)}$ = 0.7), rest to dummies	a bit of miss-classification of HportScan as Benign on 80-20 splitting
4	Add Transformed numerical features (logs, pt, qt)	same as Test 1
5	Add Binary features from ip address lib ($PCC_{(some\ binaries,\ target)}$ = -(0.1--0.2))	same as Test 1
6	Disentangle horizontal from vertical scans - 1st correction	~same as Test 1
7	Disentangle horizontal from vertical scans - 2nd correction	high miss-classification for ~all classes

Test 1



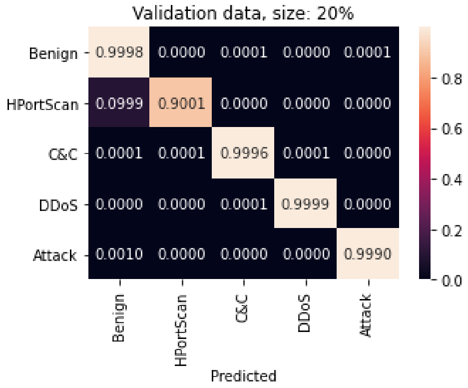
- IP addresses to octets
- Categorical as dummies

Test 2



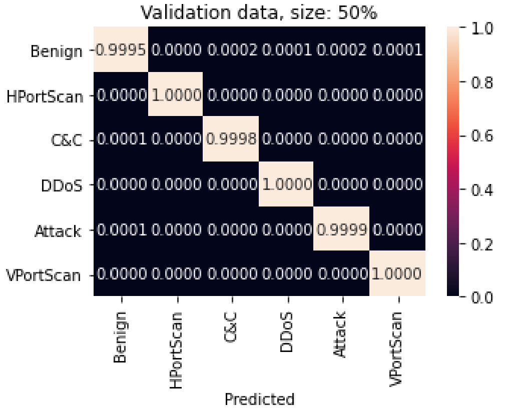
- IP addresses to octets
- Categorical as enumerated

Test 3



- IP addresses to octets
- History as enumerated, rest as dummies

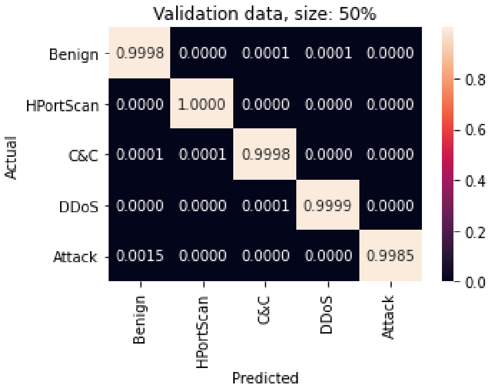
Test 6



- IP addresses to octets
- Categorical as dummies
- Disentangle port scans - 1st correction

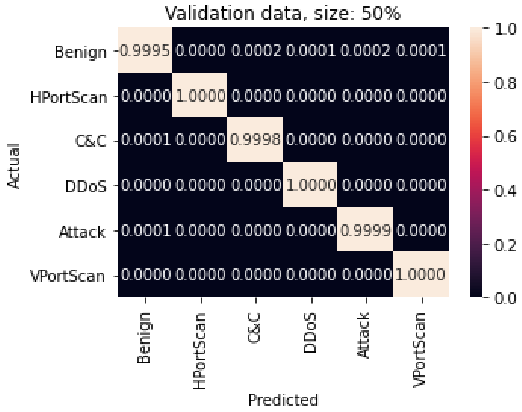
Disentangling horizontal from vertical scans – 1st correction

Test 1



- IP addresses to octets
- Categorical as dummies

Test 6



- IP addresses to octets
- Categorical as dummies
- Disentangle port scans - 1st correction

By disentangling the horizontal from the vertical scans only with the 1st correction:

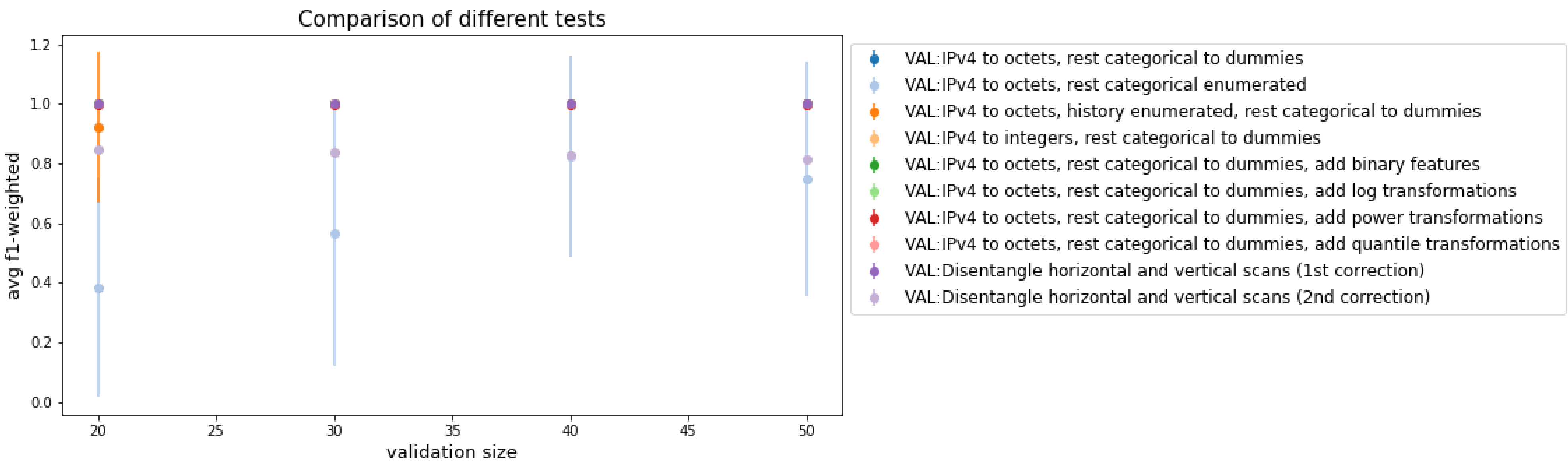
- No confusion btw vertical & horizontal port scans
- DDoS miss-classification improved to maximum
- Attack entries miss-classified as benign were reduced

improvement of malicious traffic

- but a bit of increase of benign entries miss-classified as C&C, Attack and vertical port scans

This correction is helpful if one wants to catch up all the malware that runs on a network traffic and does not bother to have benign entries miss-classified as malware and needs to check them.

Comparison of Tests over EDA & Feature Engineering



- IP addresses to octets
- Categorical as dummies
- Disentangle port scans - 1st correction

reached highest f1-weighted score (~1) across all different splittings

Continue with:

- this encoding and correction
- train-validation splitting 60-40 with 10 random state

Feature Reduction

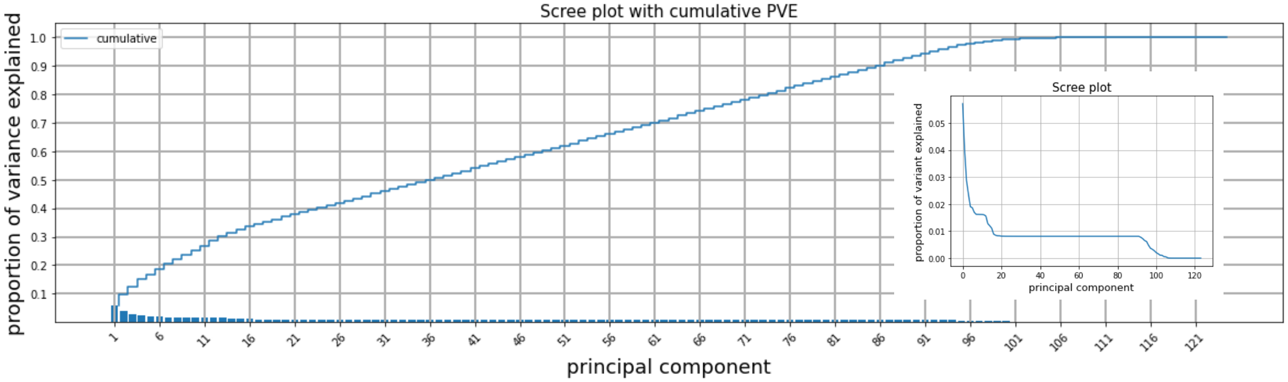
- Since after pre-processing there are more than 100 features, the **PCA** and the **SelectKBest** algorithms were explored in order to decide for a potential reduction of the them.
- For both algorithms, the features are scaled with the **Min-Max** and **Standard scalers.**
- The classifier used to select the number of top features is again the Random Forest with the same parameters mentioned earlier.
- The pipeline is predefined with the steps for testing initially set to None and changed accordingly to the tests performed.

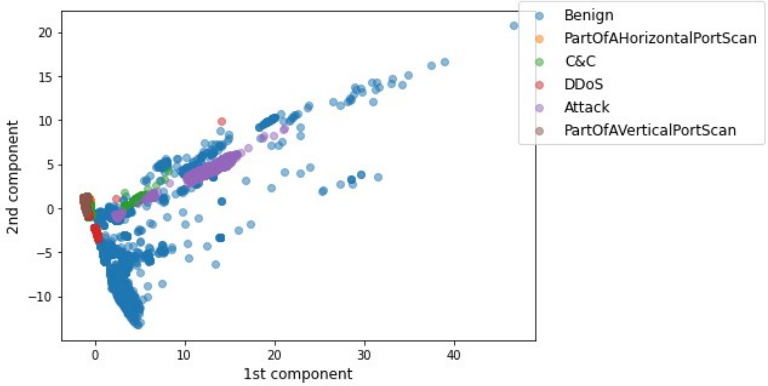
```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier


# Define the pipeline
pipe = Pipeline([('recover_nulls', pre.RecoverNansPreprocessor()),
                 ('cleaning_preprocessor', pre.CleaningPreprocessor()),
                 ('categorical_preprocessor', pre.CategoricalPreprocessor(cols_to_dummies = ['protocol', 'conn_state', 'history'])),
                 ('ip_encoding', pre.IPEncodingPreprocessor(ip_to_octets = True)),
                 ('scaler', None),
                 ('ft', None),
                 ('clf', None)

])

# Define the classifier
rf_clf = RandomForestClassifier(n_estimators=10, class_weight='balanced_subsample',
                                max_features=None, bootstrap=False, random_state=0)
```

Steps for testing

PCA

Standard Scaler

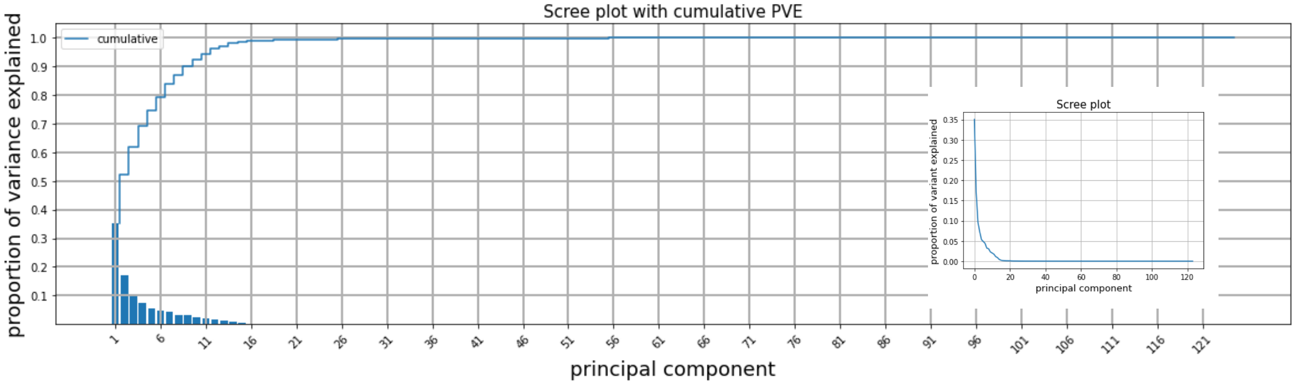


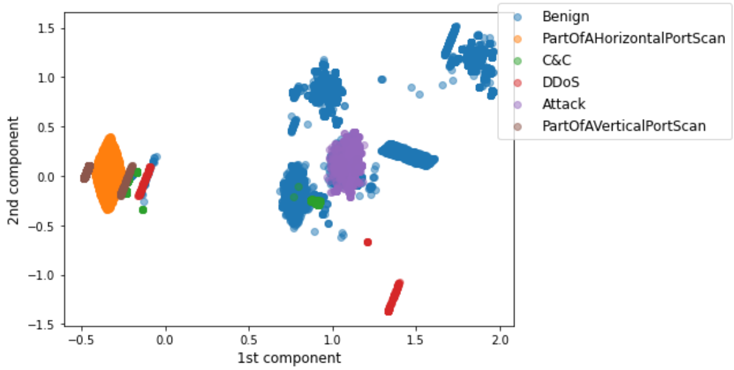


- The cumulative pve reaches about ~95% with ~95 principal components.

- With only the first 2 pr. components → pve<10%
- benign traffic not distinguishable from malicious

MinMax Scaler



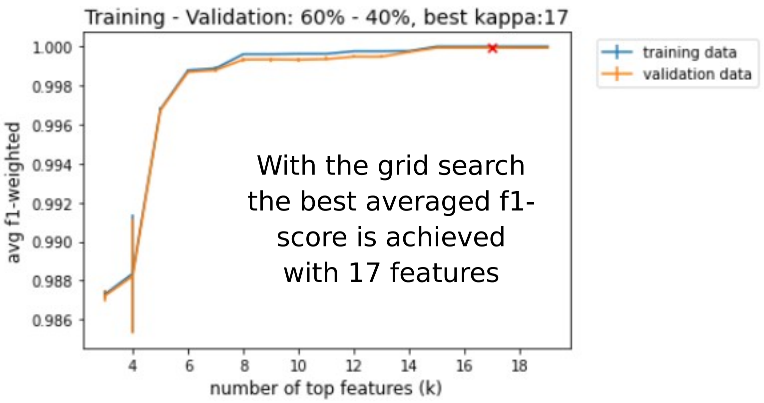


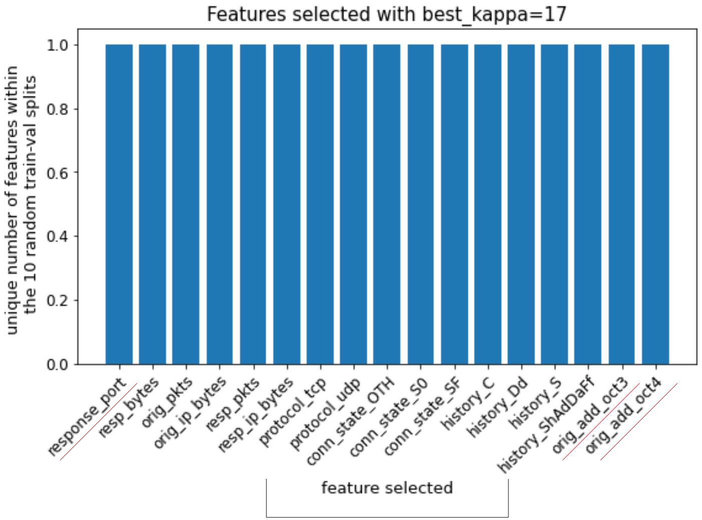
- With only ~16 pr. components we have about ~98% of the variance in the data explained.

With the first 2 p.c, there are clusters of benign entries and about half of the DDoS entries that are quite distinguishable from the other classes.

Feature Reduction

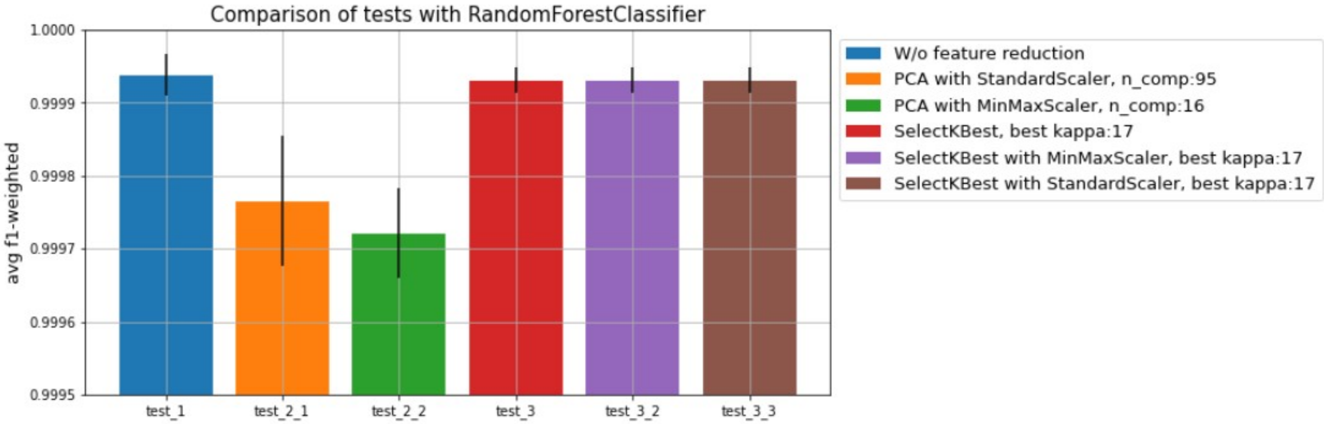
SelectKBest





#Binary features= 9

PCA & SelectKBest comparison



- The difference between the dimensionality reduction and the feature selection is subtle.
- Data do not need to be scaled before using the SelectKBest
- For the final preprocessing step → SelectKBest

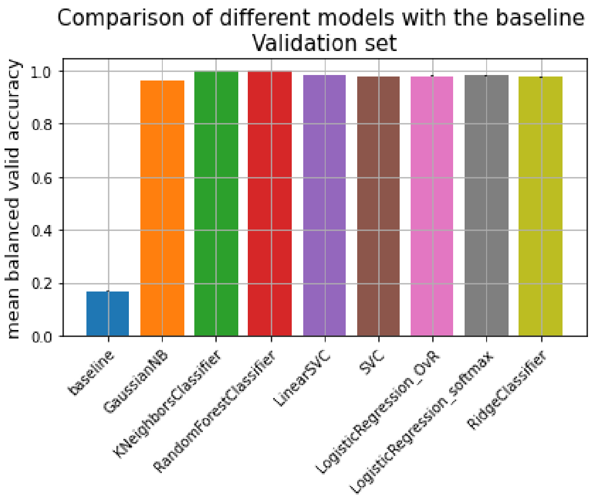
Tests with Different Models w/o fine tuning

1) Baseline → *dymmy classifier with uniform strategy*
2) Gaussian Naive Bayes → *since binary features are only 7 out of the 17*
3) Kneighbors Classifier
4) Random Forest Classifier → *bootstrap flag is on, max_features to default ='sqrt'*
5) SVC: LinearSVC, SVC with rbf kernel → *strategy for multiclass: one-vs-rest*
6) Logistic Regression one-vs-rest and softmax
7) Ridge Classifier

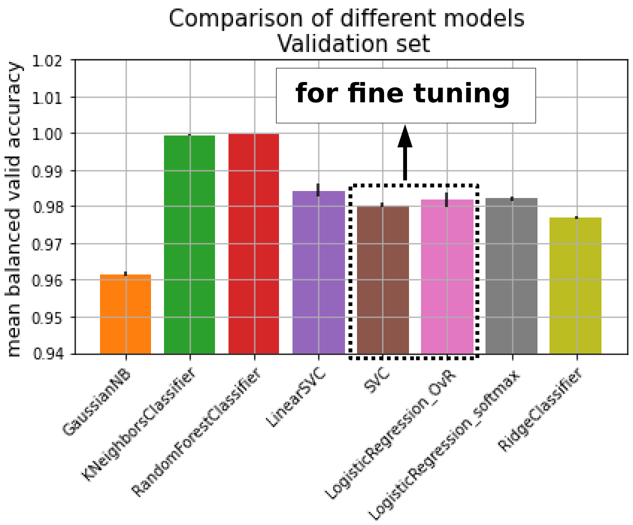
In models 4-7 there is a weight parameter for the classes to tackle imbalancing.

```python
# Define the pipeline
pipe = Pipeline([('recover_nulls', pre.RecoverNansPreprocessor()),
                 ('cleaning_preprocessor', pre.CleaningPreprocessor()),
                 ('categorical_preprocessor', pre.CategoricalPreprocessor(cols_to_dummies = ['protocol', 'conn_state', 'history'])),
                 ('ip_encoding', pre.IPEncodingPreprocessor(ip_to_octets = True)),
                 ('ft', SelectKBest(k=17)),
                 ('scaler', MinMaxScaler()),
                 ('clf', model)

])
```

Comparison of Different Models



zoom w/o baseline



~same for both training and validation sets

	Miss-classification [%]		
Model	**Benign as C&C**	**Benign as H. Port Scans**	**Benign as H. Port Scans**
Gaussian NB	20	-	-
LinearSVC	7	-	-
SVC, kerbel=RBF	10	-	1
Logistic Regression ovr	7	-	1
Logistic Regression softmax	8	-	1
Ridge Classifier	10	2	1

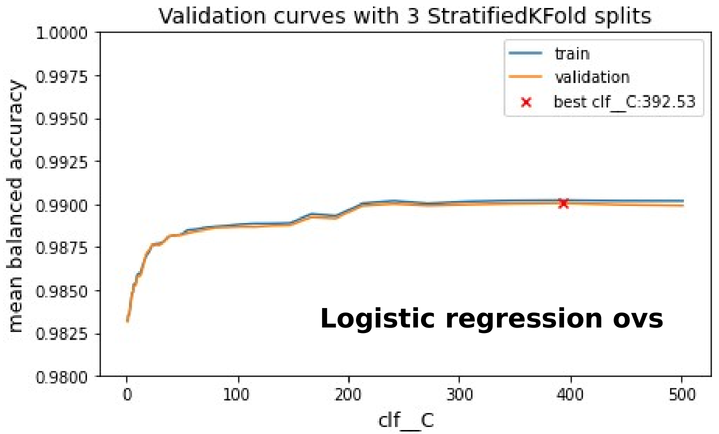
Mean balanced accuracy:

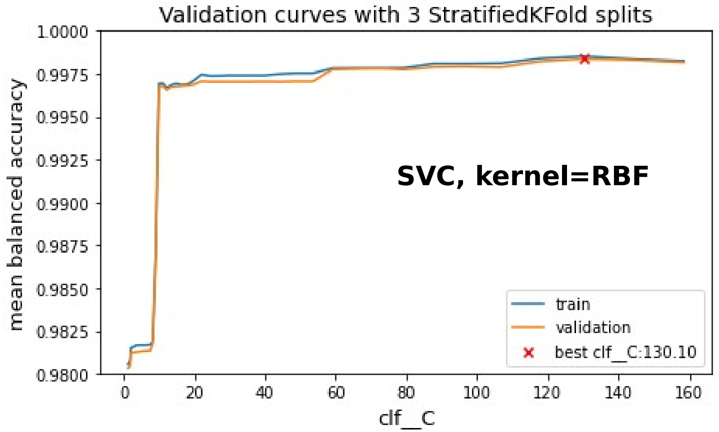
- Kneighbors and Random Forest ≈ 100%
- LinearSVC ≈ 98.5%
- SVC_rbf, Logistic Regressions ≈ 98%
- Ridge ≈ 97.5%
- GaussianNB ≈ 96%

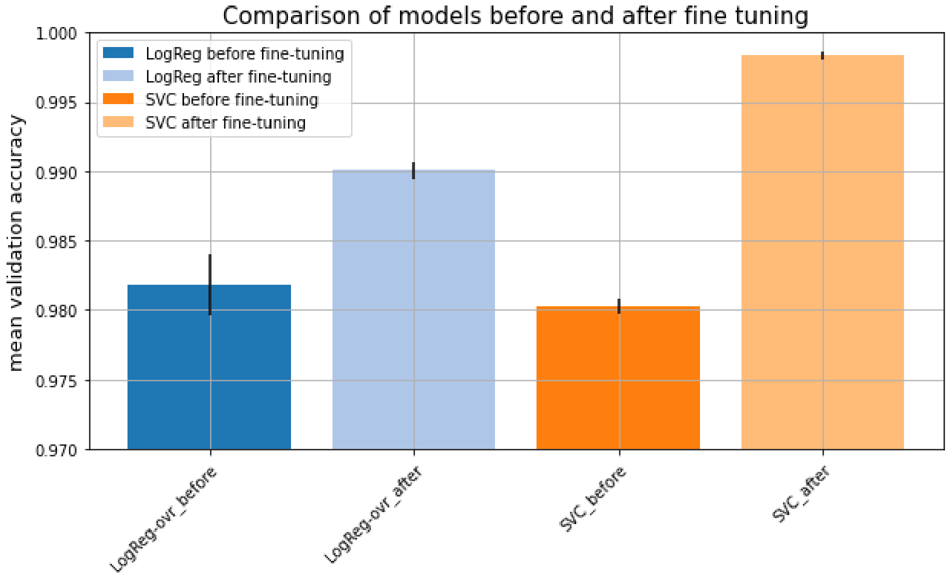
In all of them, miss-classification of Benign entries as C&C

Fine Tuning

- Fine Tuning of regularization strengths performed using the **GridSearchCV**
- Training sample is split into 3 folds using the **StratifiedKFold** cross-validator in order to preserve the percentage of samples for each class





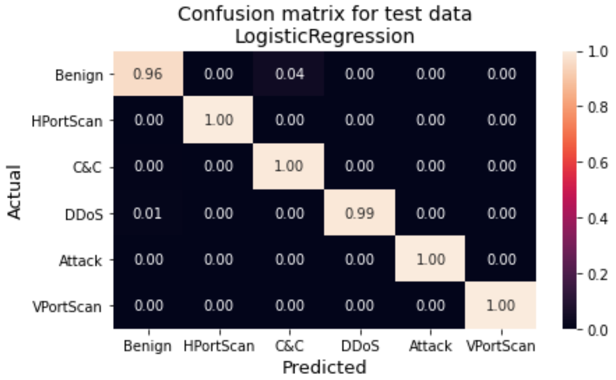


After fine tuning, the mean validation accuracy was increased for both models:

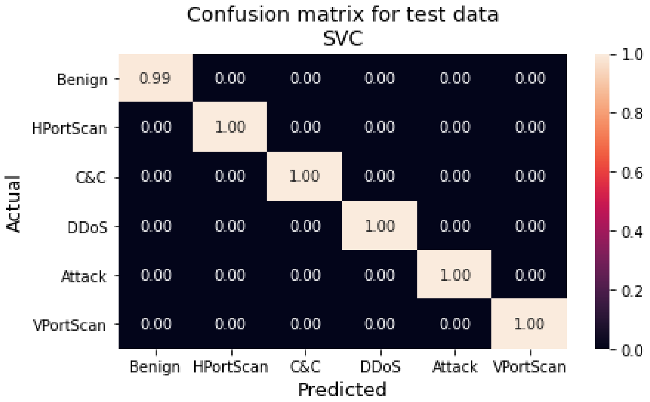
- ~1% for the Logistic Regression
- ~1.5% for the SVC

Evaluation on Test Data

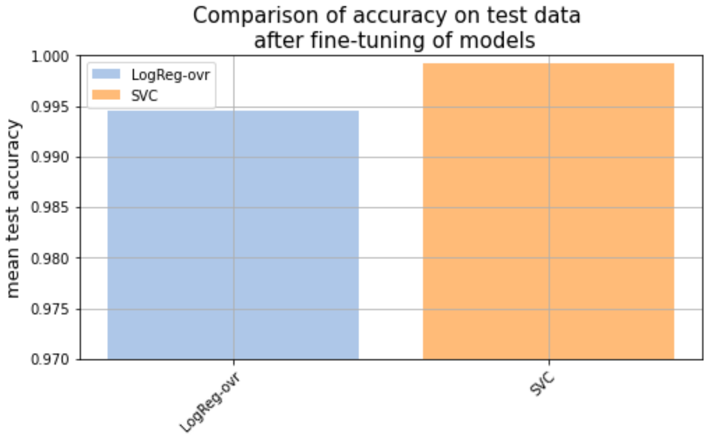
The classifiers with the best regularization strengths found were evaluated on the test data



- 4% of Benign entries miss-classified as C&C
- no miss-classification of C&C as benign, as was the case before fine tuning



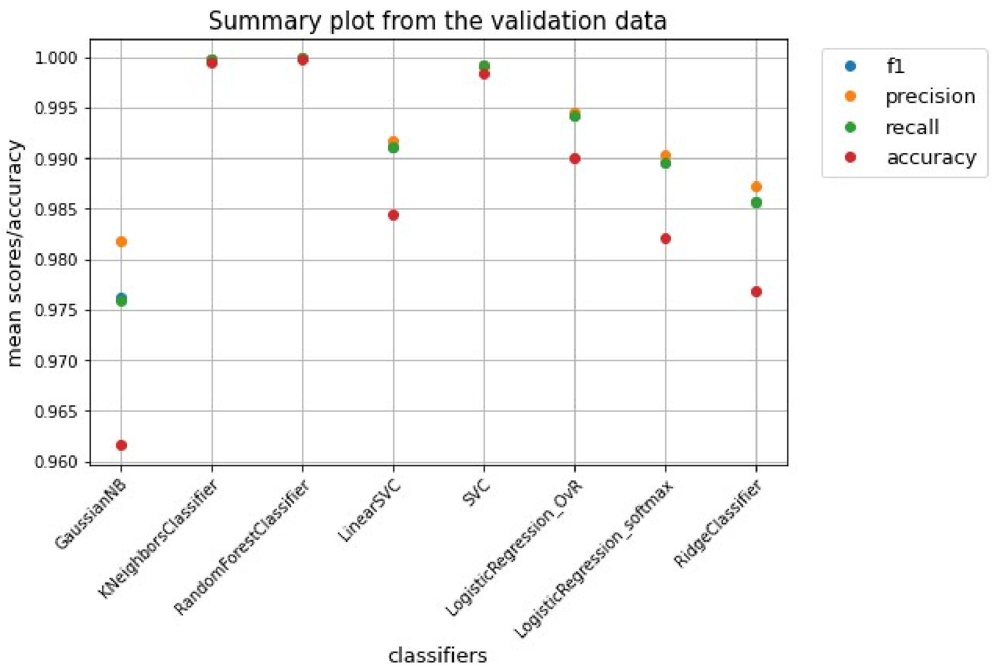
Only 1% of Benign entries miss-classified as C&C



Mean test accuracy:

- ~99.5% for the Logistic Regression ovr
- ~99.9% for the SVC with RBF kernel

Comparison of all Models



The results shown here are after finding the best regularization strength for the Logistic Regression ovr and the SVC

The overall performance of all classifiers except the GaussianNB is >97.5% with the KNeighbors & the RandomForest classifiers, as well as the SVC with rbf kerner reaching excellent scores and accuracy close to 1. While the GaussianNB reaches a mean accuracy of 96%.

The fact that the accuracy for all models except the Kneigbors & Random Forest Classifiers and SVC is ~1% lower than the scores is because of the miss-classification of the Benign entries (TN) as C&C (TP) as we saw earlier.

$$\textbf{Accuracy} < \textbf{F1} \Leftrightarrow$$

$$\frac{TP+TN}{TP+TN+FP+FN} < \frac{2TP}{2TP+FP+FN} \Leftrightarrow$$

$$(TP+TN)\cdot(2TP+FP+FN) < 2TP\cdot(TP+TN+FP+FN) \Leftrightarrow$$

$$\cancel{2TP^2} + TP\cdot FP + TP\cdot FN + \cancel{2TP\cdot TN} + TN\cdot FP + FN\cdot TN < \cancel{2TP^2} + \cancel{2TP\cdot TN} + 2TP\cdot FP + 2TP\cdot FN \Leftrightarrow$$

$$TN\cdot FP + FN\cdot TN < TP\cdot FP + TP\cdot FN \Leftrightarrow$$

$$TN\cdot\ (FP+FN)\ < TP\cdot\ (FP+FN)\ \Leftrightarrow$$

$$\textbf{TN} < \textbf{TP}$$

Conclusions and Further Discussion

- As we saw, the features provided by the monitoring system of the network are enough to reach a very good accuracy with different supervised models even without fine tuning.

- There are of course other models that could be tried like for e.g. other ensemble methods such as BaggingClassifier, ExtraTreesClassifier etc or Dense Networks and even Neural Networks.
  - One could also try to downsample the majority class and check if for example the Naive Bayes improves.

- In general though, I think it would be more interesting as a next step to approach the anomaly detection in this kind of networks in a semi-supervised way, because examining all the log files of the monitoring system and labeling them requires quite some time and manpower.

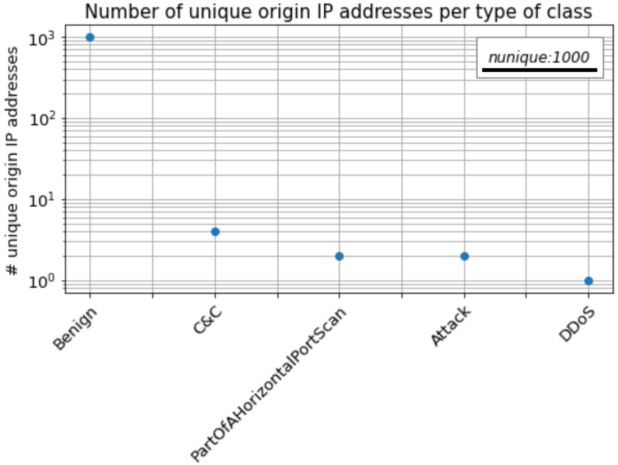
Thank you!

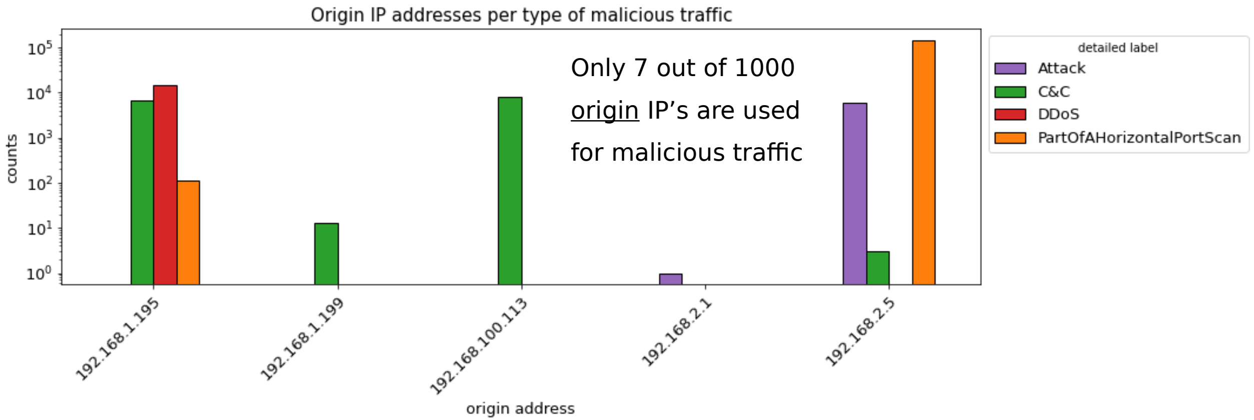
Additional Slides

EDA: Categorical Features

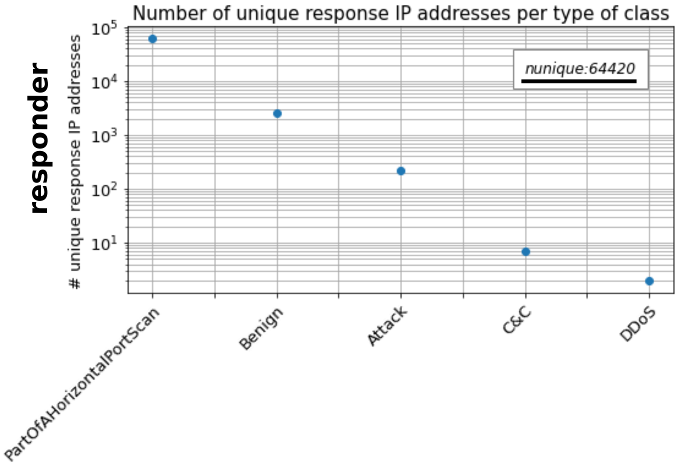
IP addresses:

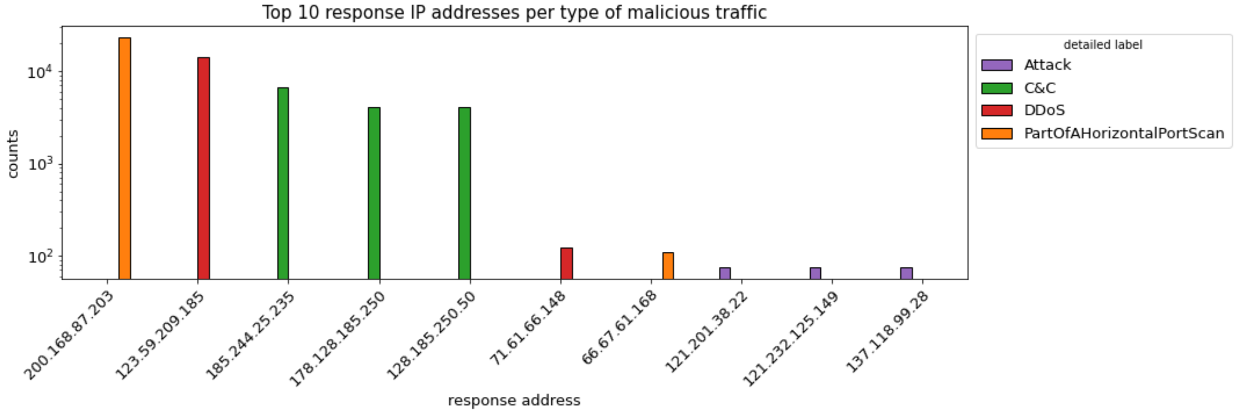
originator





responder

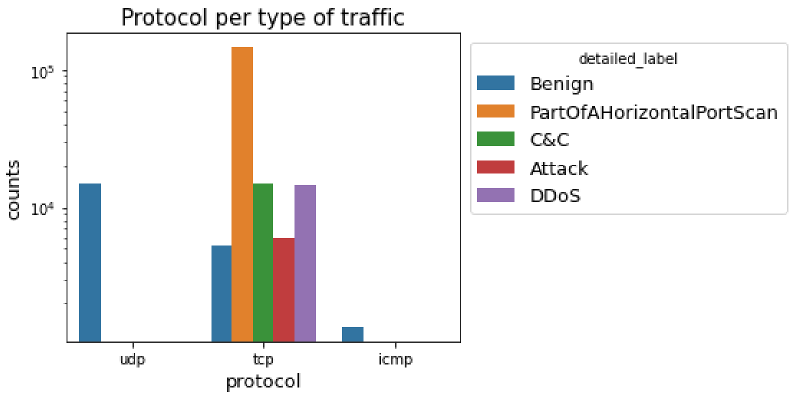




EDA: Categorical Features

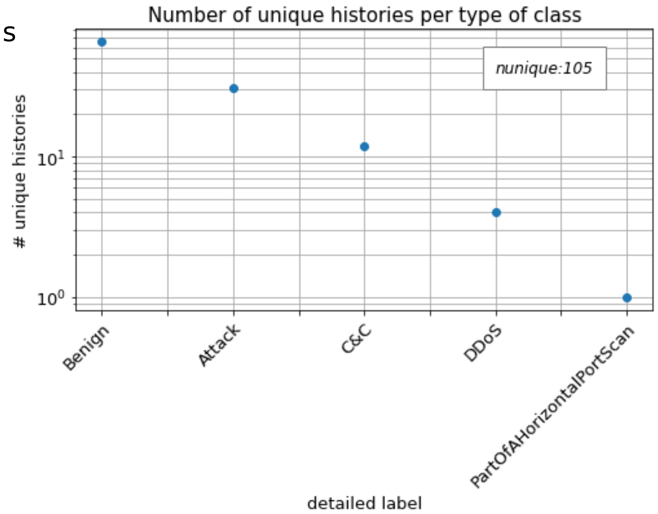
Protocol:

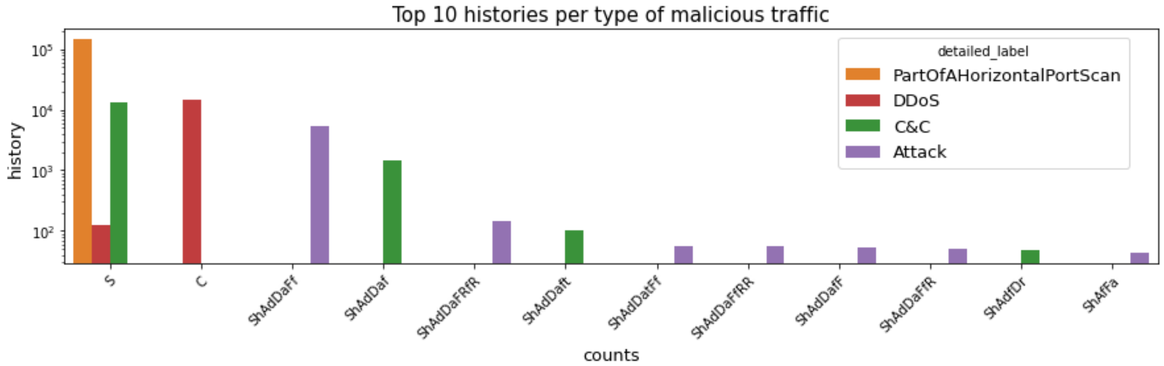
- 3 unique values:
  - TCP → **all type of traffic**
  - UDP, ICMP → **only benign**



History:

- represents the state of protocols used to perform connections between devices or servers
- encoded as single letter or combination of letters
- 105 unique values
  - **Benign**:~65
  - **Attack**: 30
  - **C&C**: ~11
  - **DDoS**: 4
  - **PartofAHorizontalScan**: 1

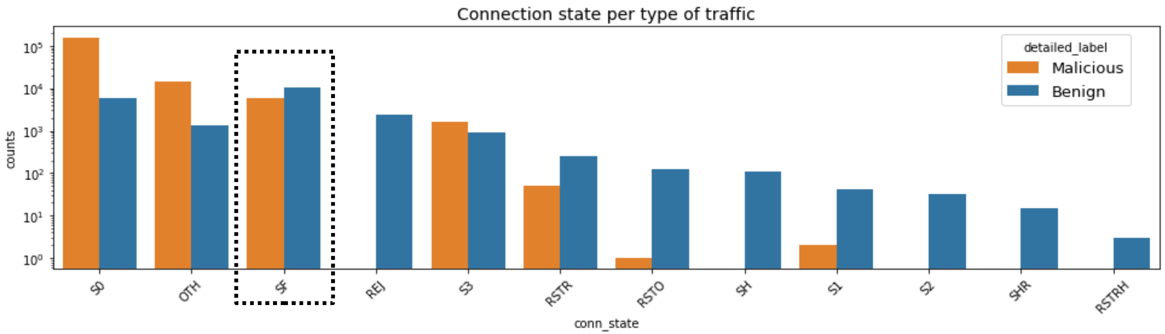


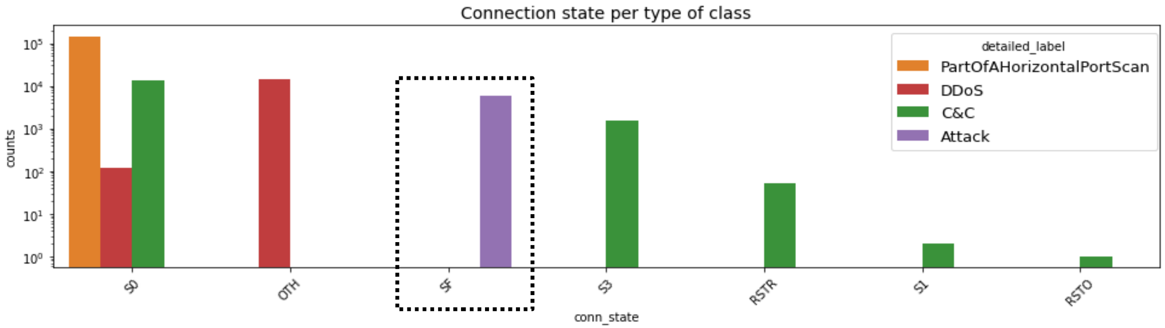


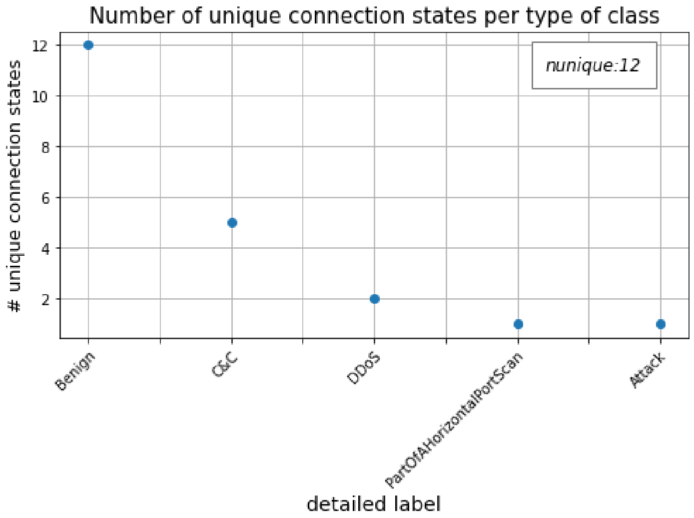
EDA: Categorical Features

Connection state:

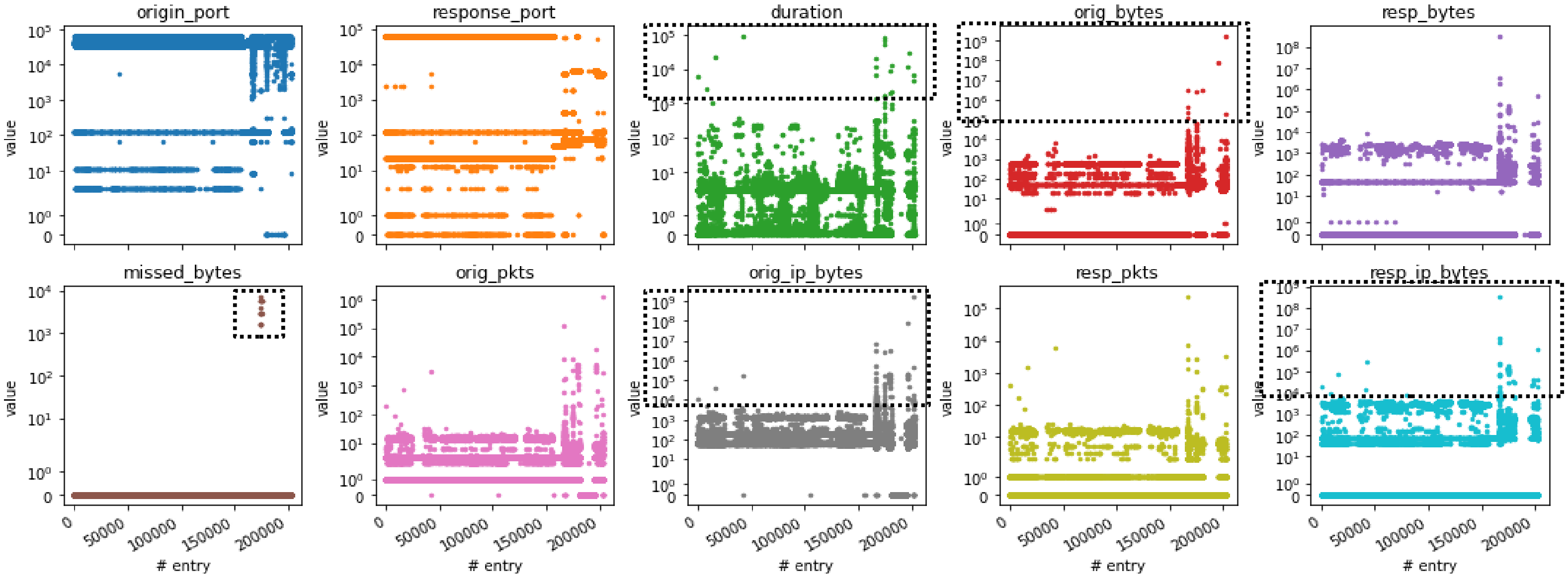
- 12 unique values:
  - only 1 indicates a normal establishment and termination (**SF**) while the rest indicate a specific problem during the connection.
  - all of them used by benign traffic
  - 7 used by malicious traffic







EDA: Numerical Features



Outliers - Case 1:

```
data_df.loc[(data_df['duration']>500) |
            (data_df['orig_bytes']>=1e5) |
            (data_df['orig_ip_bytes']>=1e4) |
            (data_df['resp_ip_bytes']>=1e4)][[
```

```
 --------------------------------
Total number of entries: 98
 --------------------------------
```
→ **to be dropped**

Outliers - Case 2:

```
data_df.loc[data_df['missed_bytes']>1]
```

```
 ------------------------------------------------
 Missing bytes:
 ------------------------------------------------
detailed_label  protocol  conn_state
Benign          tcp       SF              18
dtype: int64
 ------------------------------------------------
Total number of entries: 18
 ------------------------------------------------
```

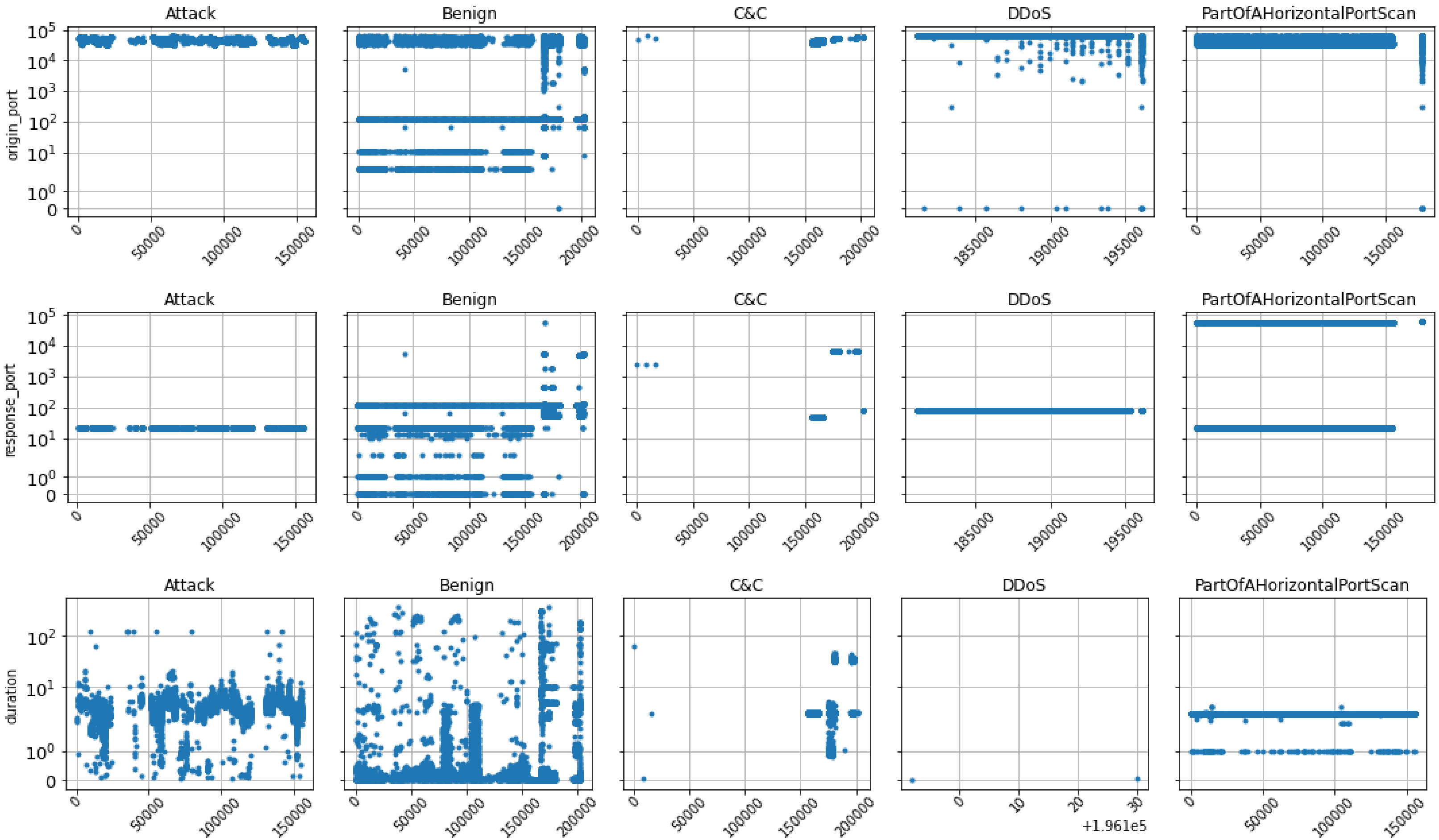
Since:

- the TCP protocol ensures re-transmission of data packets and
- the connection state is normal establishment and termination

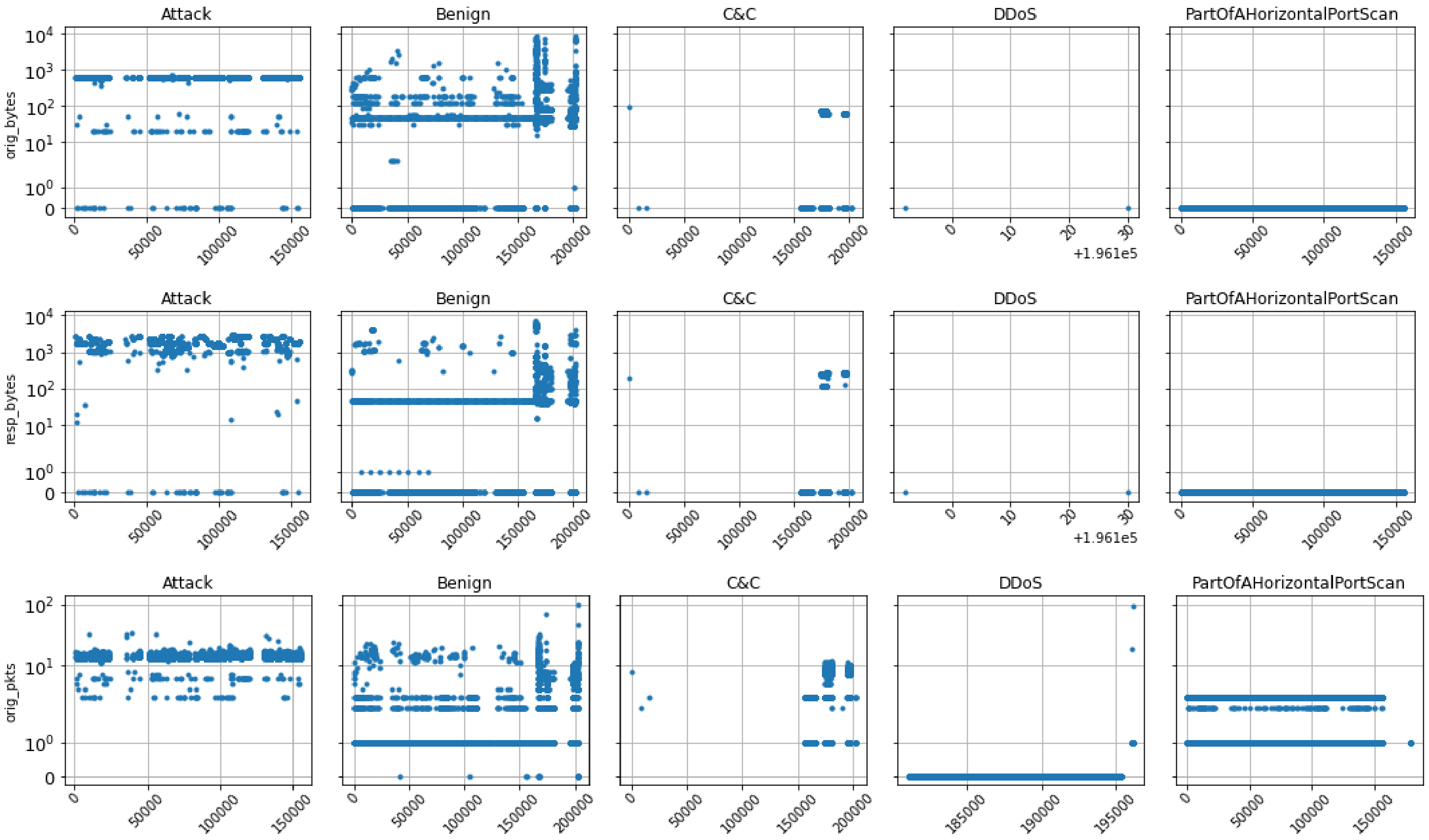
there shouldn't be any missing bytes.

Feature can be dropped as all the rest of values are only 0.

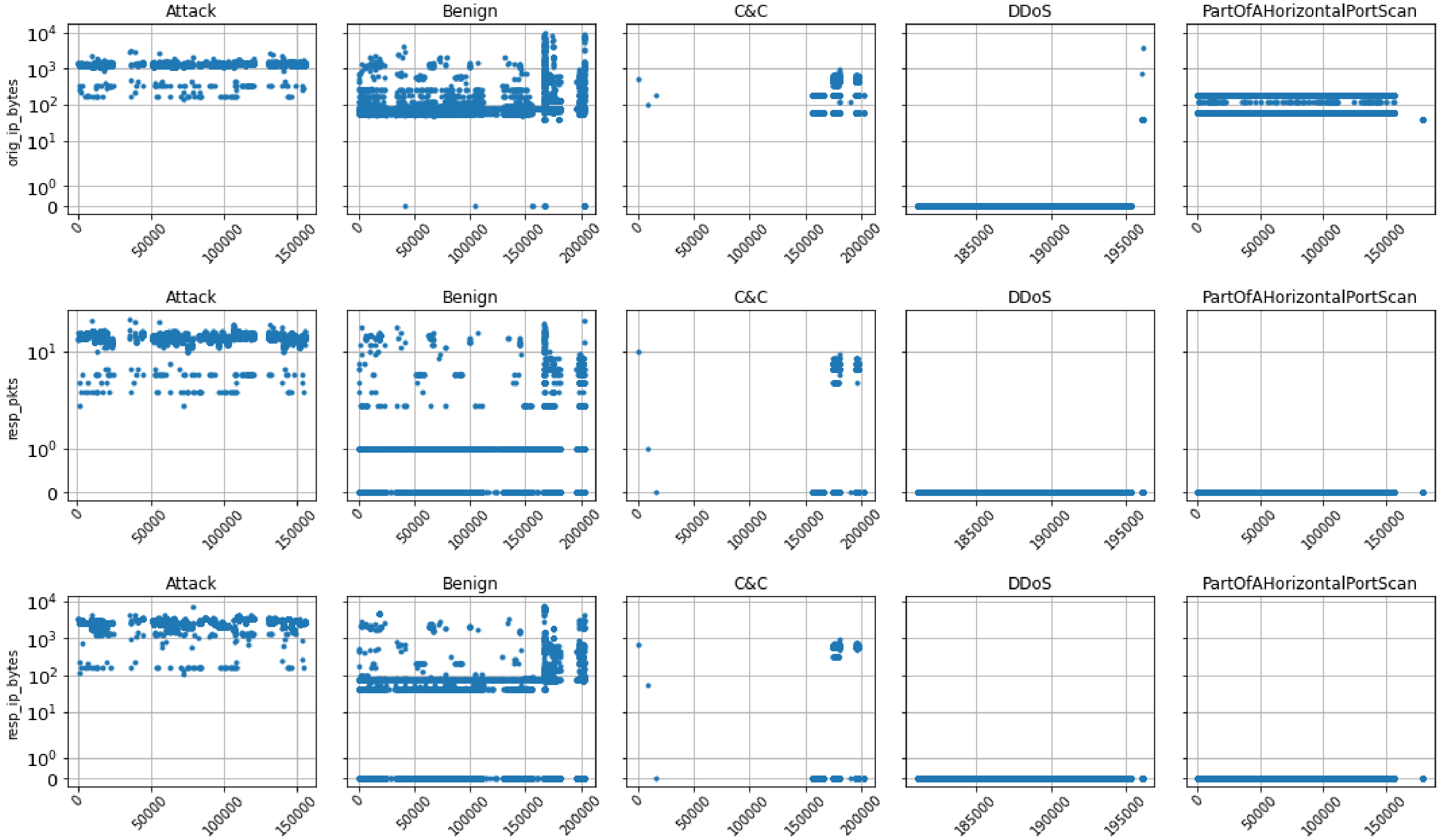
EDA: Numerical Features



EDA: Numerical Features

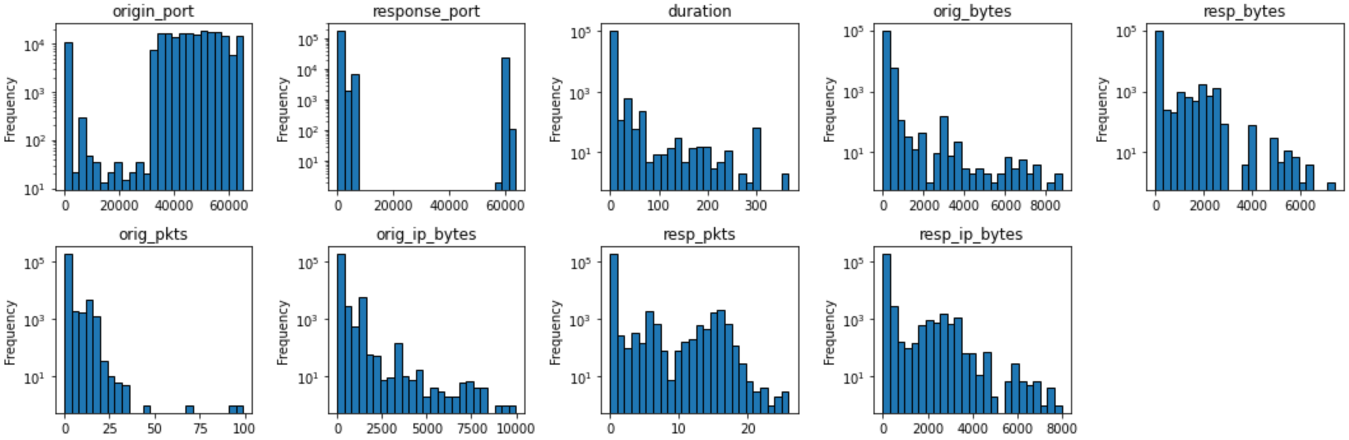


EDA: Numerical Features

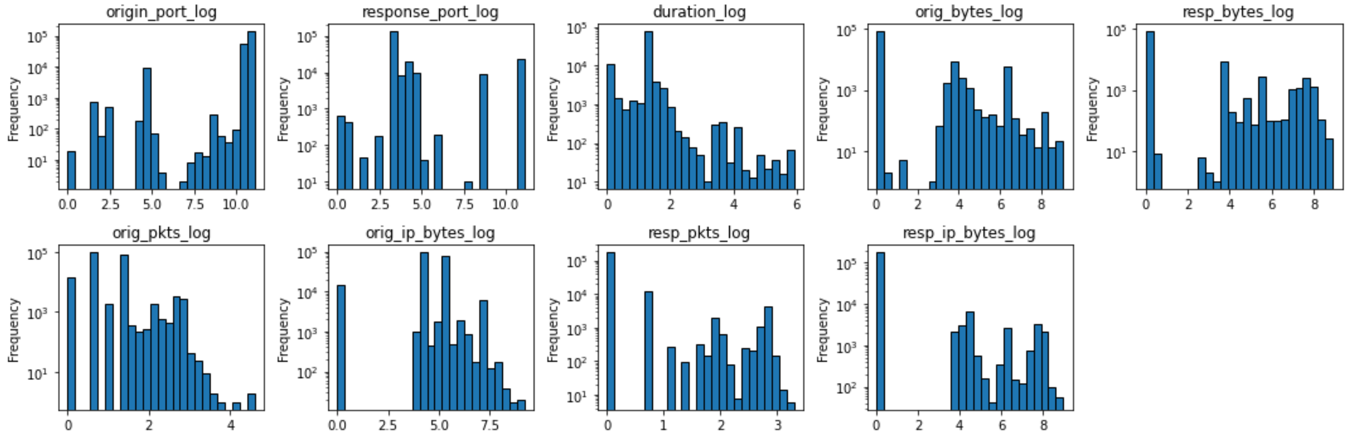


EDA: Feature Engineering

- **Untransformed features:**

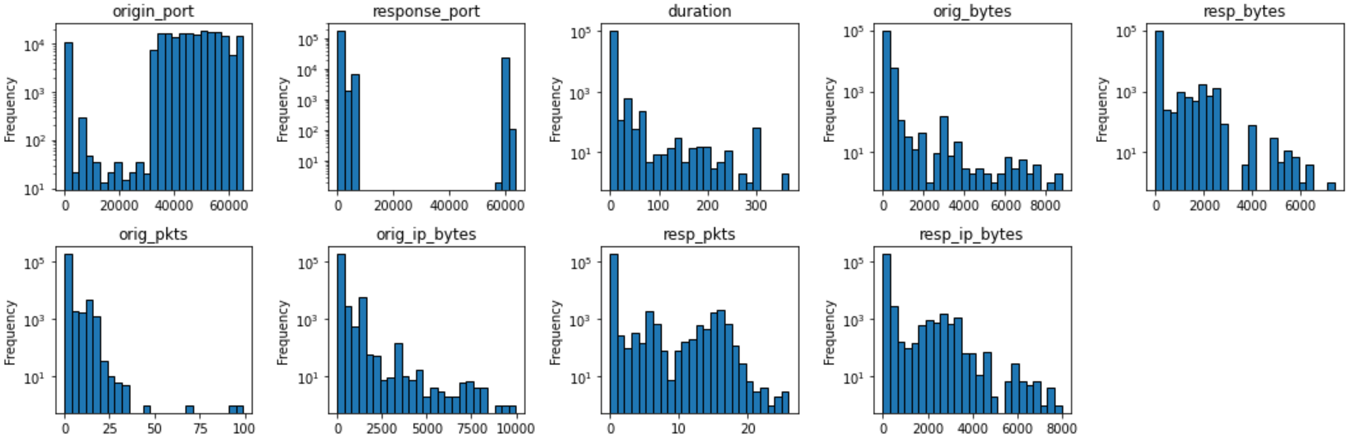


- **Log transformed features:**

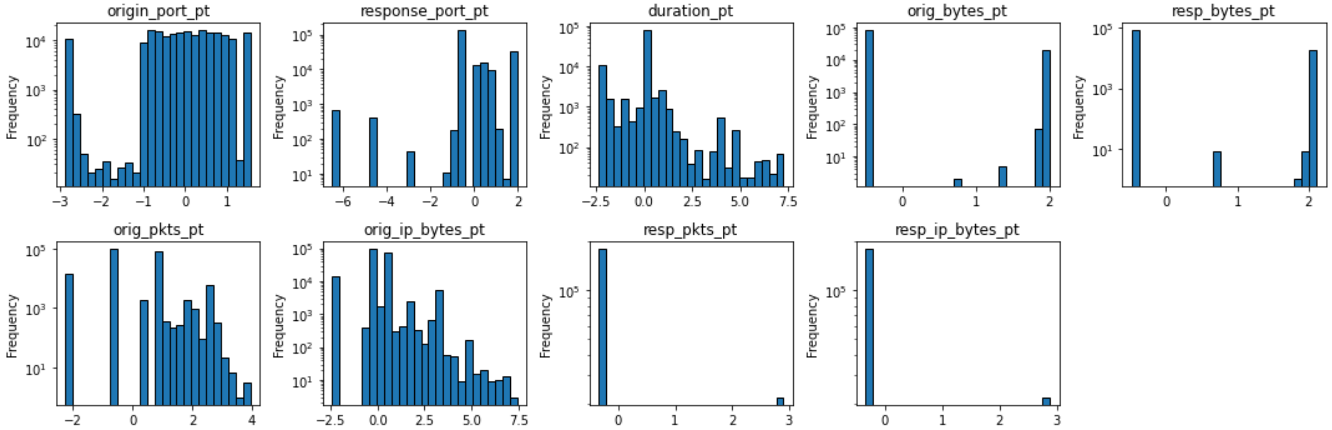


EDA: Feature Engineering

- **Untransformed features:**

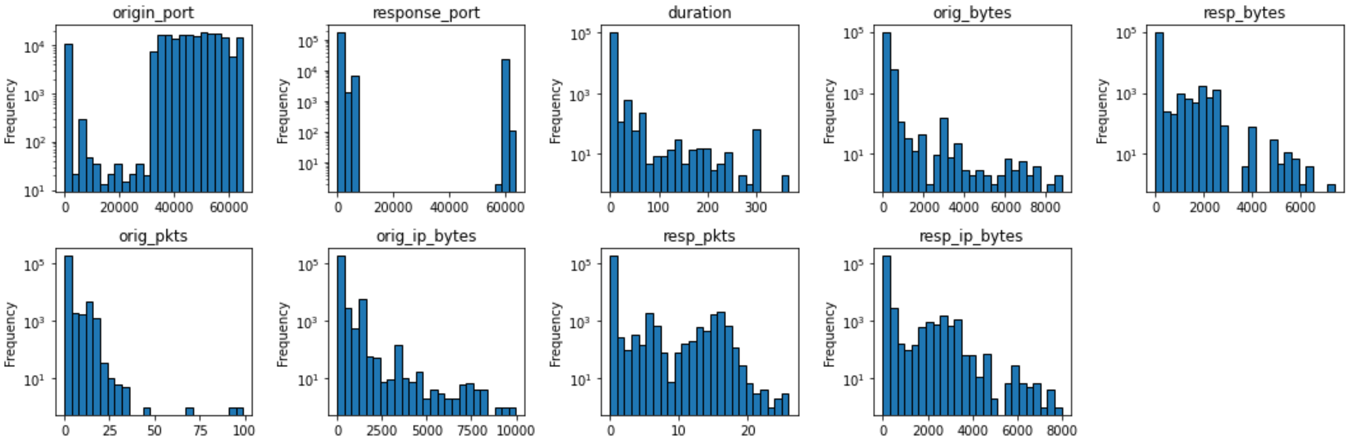


- **Power transformed features:**

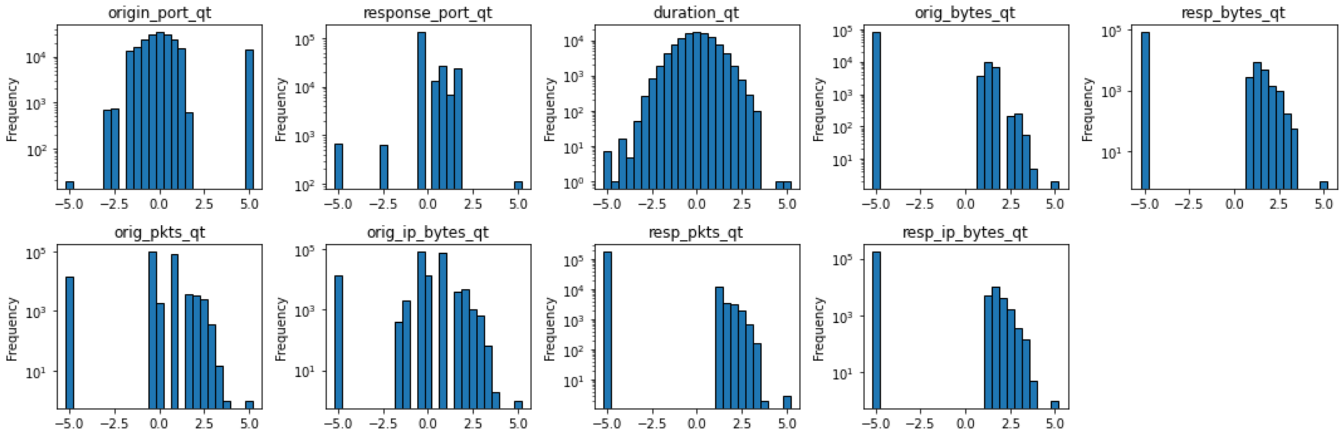


EDA: Feature Engineering

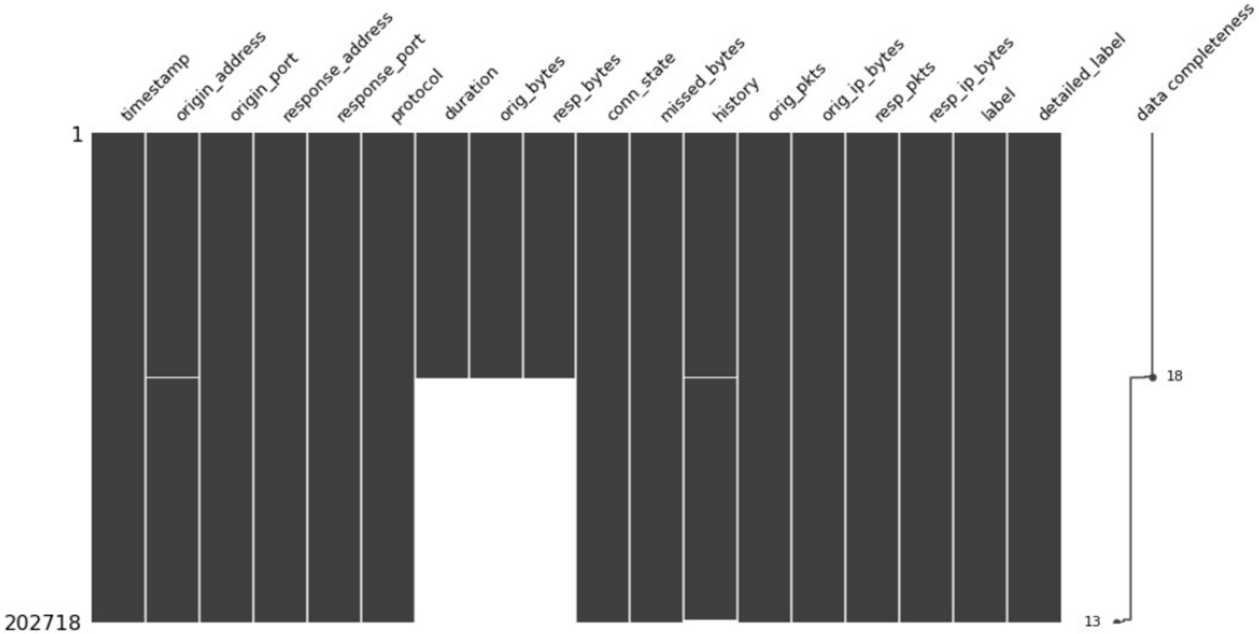
- **Untransformed features:**



- **Quantile transformed features:**



EDA: Missing values

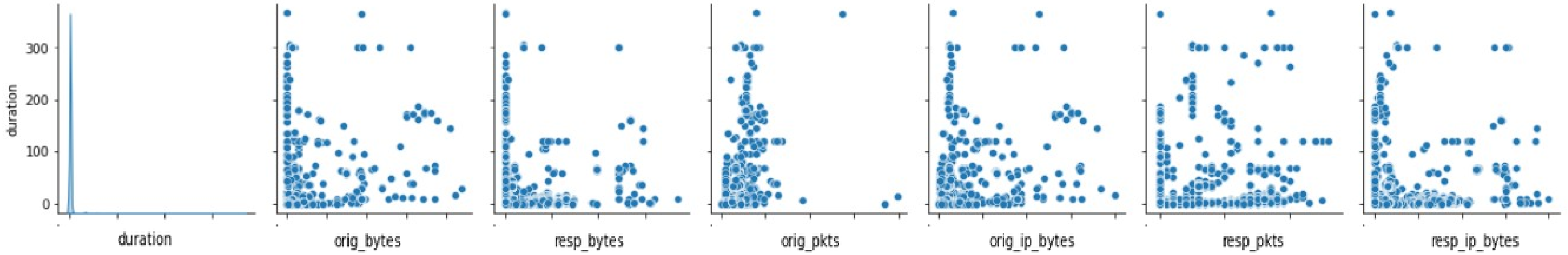


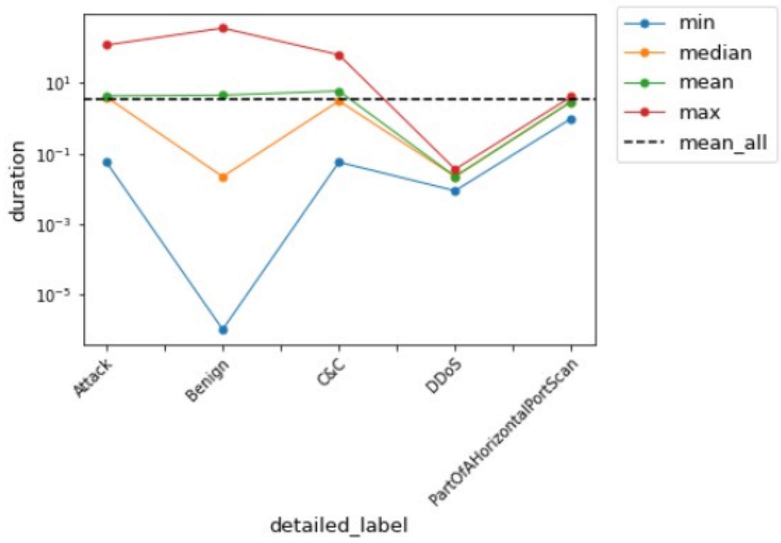
missingno lib

- **origin_address:** missing only IPv6 (211 entries in total from benign traffic) → can be replaced by the most common used or dropped.
- **history:** missing only from benign traffic using the ICMP protocol. Since the history represents only the state of protocols used to perform connections between devices or servers, like the TCP or UDP protocols → can be replaced by “no_history”
- **duration**, **orig_bytes** & **resp_bytes:** missing values from same entries

EDA: Missing values

Duration:

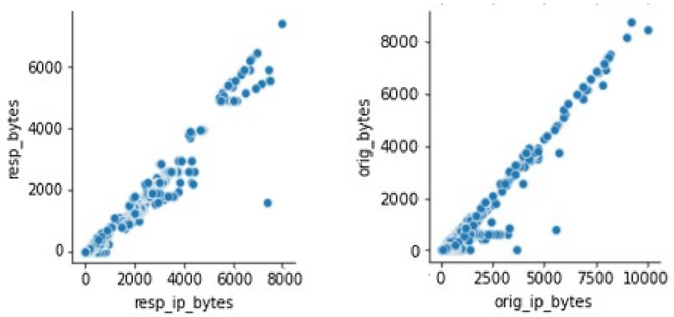




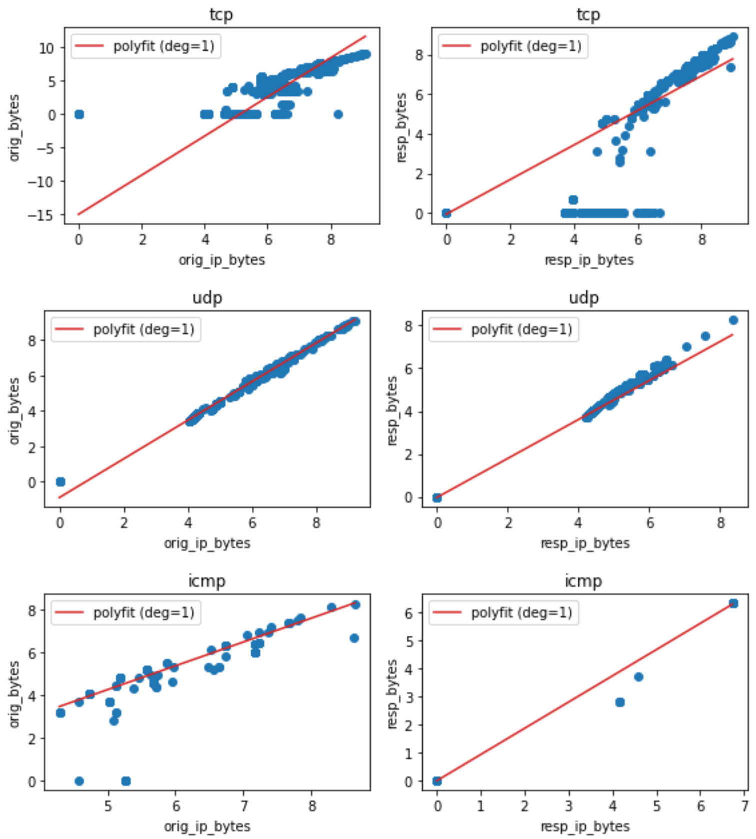
- No correlation with other numerical features
- Can be replaced by “mean_all”

EDA: Missing values

Origin and response bytes:



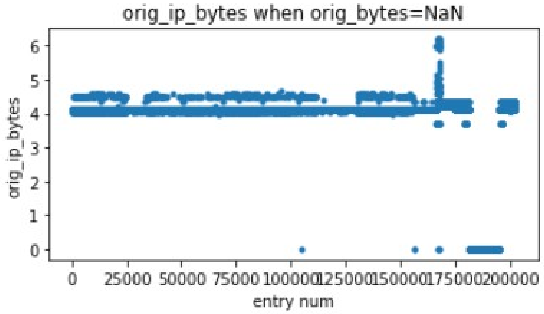
- There is a linear correlation btw bytes and ip_bytes from both originator and responder
- Should be separated according to the protocol as:
  - data transmitted through UDP are not encapsulated within an IP frame → ip_bytes = bytes.
  - data transmitted through ICMP or TCP are usually encapsulated → ip_bytes ≠ bytes.
  - not a good idea though as:
    - from originator through TCP → linearity fails and in log transformed values there are negative values
    - through ICMP there are only few entries → fit will fail after splitting the data for training, validation and test

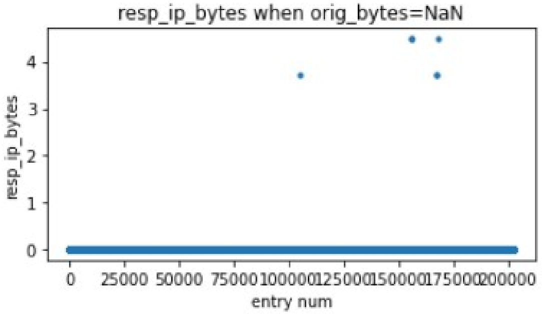


features in their log transformation

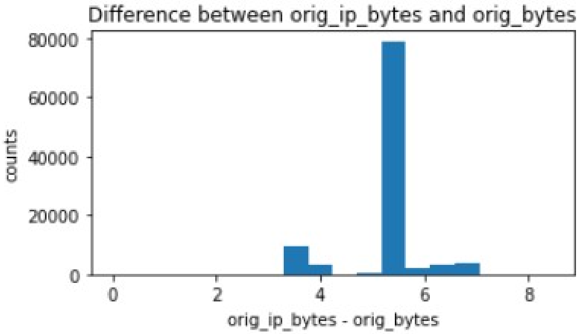
EDA: Missing values

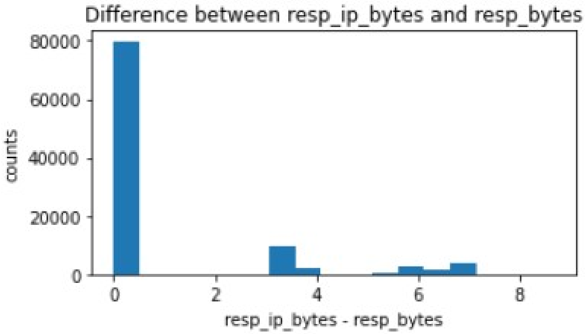
- Instead look the corresponding ip bytes (in their log transformation)





- And try to find the size of ip header and footer



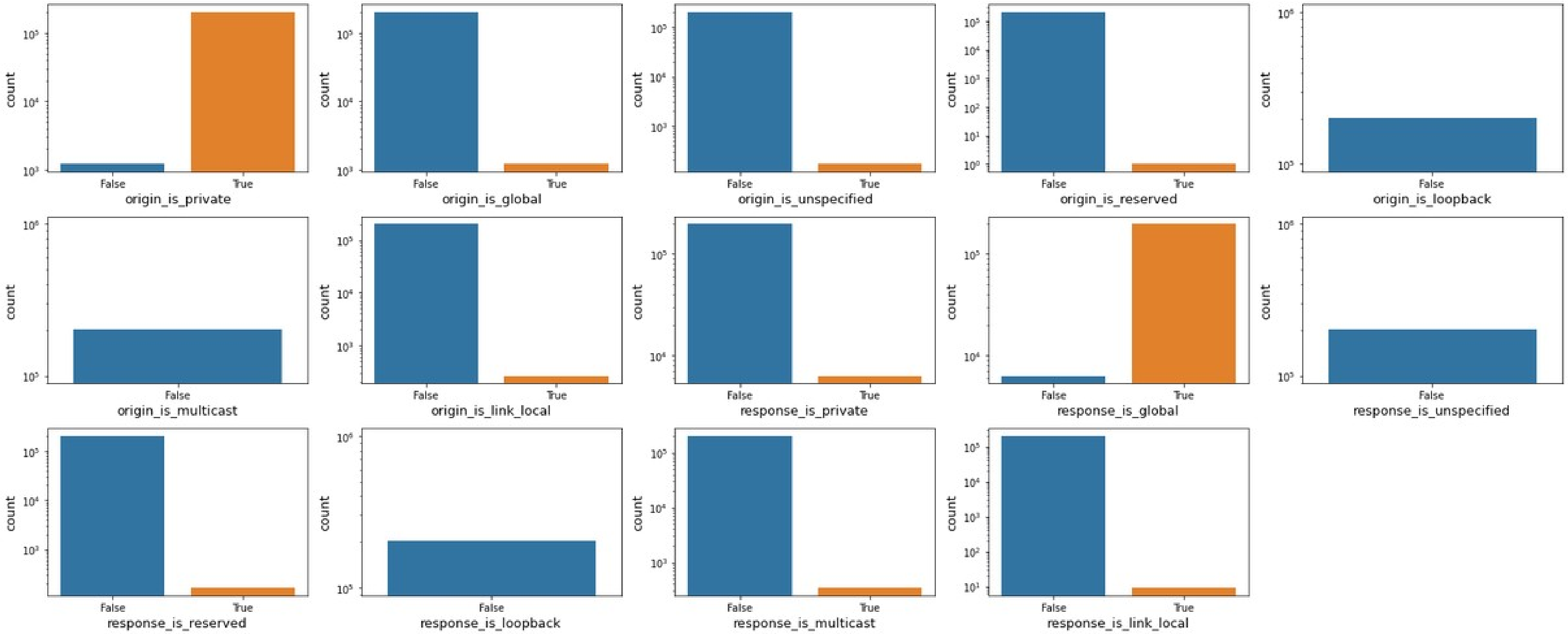


Replacement of missing values:

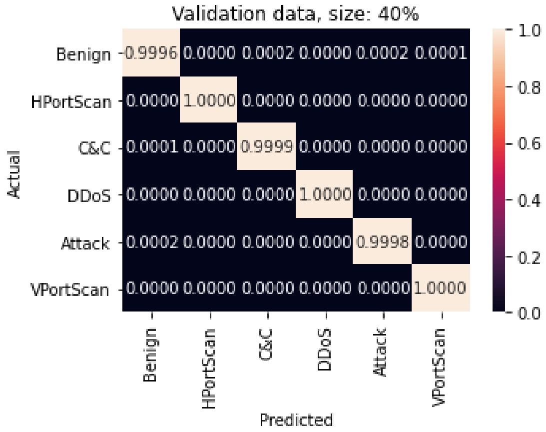
- for the originator, since the ip_bytes have values between 4-6 when the corresponding bytes are missing, instead of subtracted 5 and ending up with negative values we can fill in the missing entries with zeros.
- for the responder, we can fill in the missing values with the corresponding ip_values.

Feature Engineering

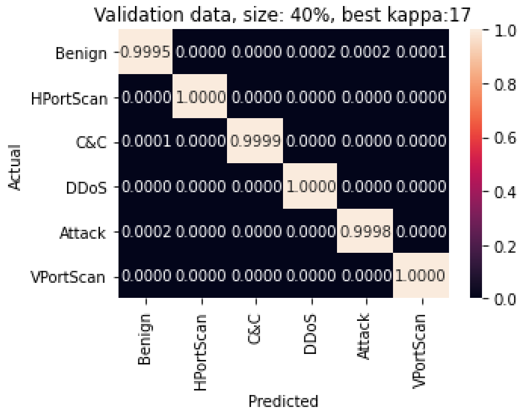
- **Binary features from ip address lib:**



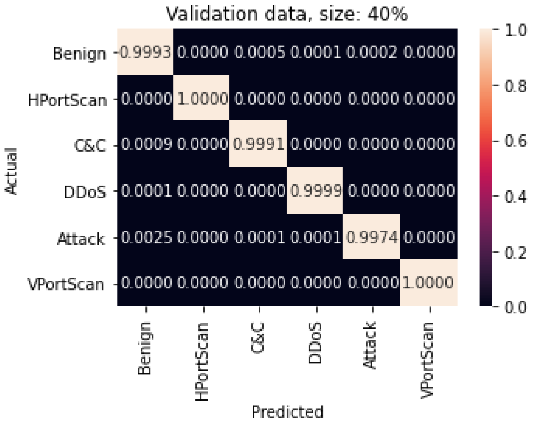
Feature Reduction – Confusion Matrices



W/o feature reduction

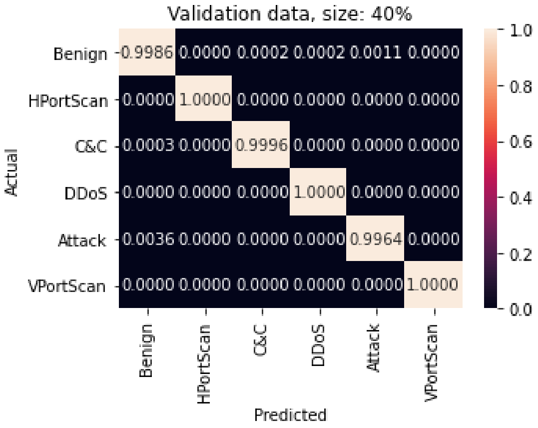


SelectKBest, k=17



PCA – Standard Scaler

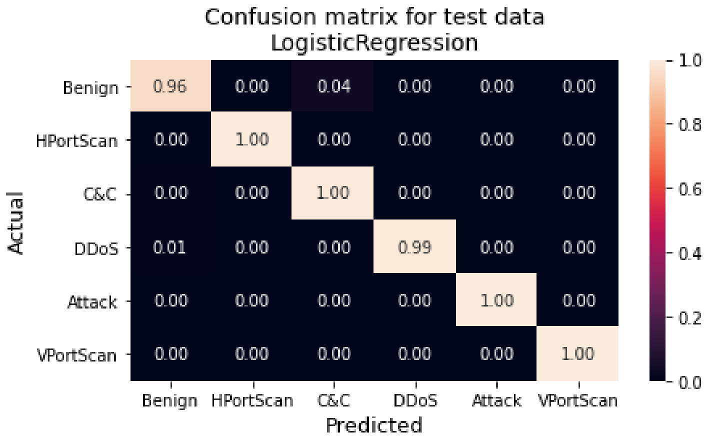
n_components = 95



PCA – MinMax Scaler

n_components = 16

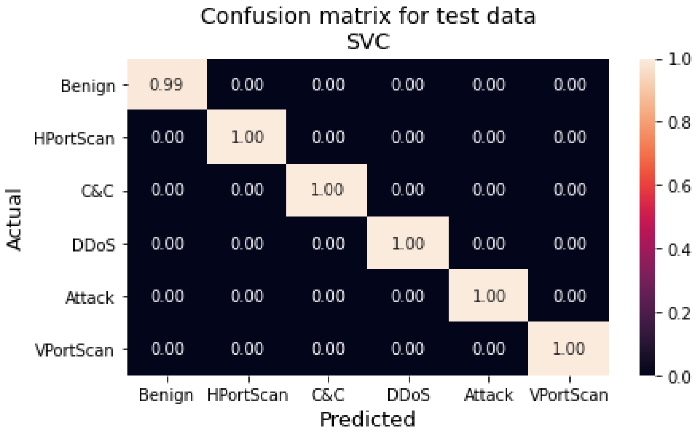
All results from fine tuning of Logistic Regression ovr



Classification report for test data (LogisticRegression)

	precision	recall	f1-score	support
Benign	0.99	0.96	0.97	6424
HPortScan	1.00	1.00	1.00	36691
C&C	0.94	1.00	0.97	4481
DDoS	1.00	0.99	1.00	4315
Attack	0.99	1.00	1.00	1788
VPortScan	1.00	1.00	1.00	7022
micro avg	0.99	0.99	0.99	60721
macro avg	0.99	0.99	0.99	60721
weighted avg	0.99	0.99	0.99	60721

All results from fine tuning of SVC with RBF kernel



Classification report for test data (SVC)

	precision	recall	f1-score	support
Benign	1.00	0.99	1.00	6424
HPortScan	1.00	1.00	1.00	36691
C&C	1.00	1.00	1.00	4481
DDoS	0.99	1.00	1.00	4315
Attack	0.99	1.00	1.00	1788
VPortScan	1.00	1.00	1.00	7022
micro avg	1.00	1.00	1.00	60721
macro avg	1.00	1.00	1.00	60721
weighted avg	1.00	1.00	1.00	60721