Francesco Venturini, No. 548415

Artificial Intelligence For Cybersecurity

Classification for the UNSW-NB15

The Problem

The goal of the project is to devise a general classifier that categorizes each individual sample as '*attack*' or '*normal*' network traffic.

The objective is to correctly develop a complete Knowledge Discovery from Data process (aka Data Mining).

In parallel, a study of a multi-class classification problem was done, based on 10 types of attack categories.

The work should produce good performance metrics that will be validated, evaluated and compared with existent works.

Solution Overview

1) Data Acquisition and Exploration

2) Data Cleaning

3) Data Selection and Data Preparation

4) Data Modelling (classification)

5) Validation and Evaluation

6) Results Interpretation

The Dataset

It is the UNSW-NB15, created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours.

Tcpdump tool is utilised to capture 100 GB of the raw traffic (e.g., Pcap files).

The Dataset

The Argus, Bro-IDS tools are utilised and twelve algorithms are developed to generate totally 49 features with the class label.

The total number of records is 2.540.044 which are stored in four CSV files (only a subset is used)

EDA

This is how the dataset looks like

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss	dloss	sinpk
0	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	252	254	1.415894e+04	8495.365234	0	0	24.29560
1	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	62	252	8.395112e+03	503571.312500	2	17	49.91500
2	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	62	252	1.572272e+03	60929.230470	1	6	231.87557
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	62	252	2.740179e+03	3358.622070	1	3	152.87654
4	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	254	252	8.561499e+03	3987.059814	2	1	47.75033
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
257668	82328	0.000005	udp	-	INT	2	0	104	0	200000.005100	254	0	8.320000e+07	0.000000	0	0	0.00500
257669	82329	1.106101	tcp	-	FIN	20	8	18062	354	24.410067	254	252	1.241044e+05	2242.109863	7	1	55.88005
257670	82330	0.000000	arp	-	INT	1	0	46	0	0.000000	0	0	0.000000e+00	0.000000	0	0	60000.72000
257671	82331	0.000000	arp	-	INT	1	0	46	0	0.000000	0	0	0.000000e+00	0.000000	0	0	60000.73200
257672	82332	0.000009	udp	-	INT	2	0	104	0	111111.107200	254	0	4.622222e+07	0.000000	0	0	0.00900

257673 rows × 46 columns

EDA

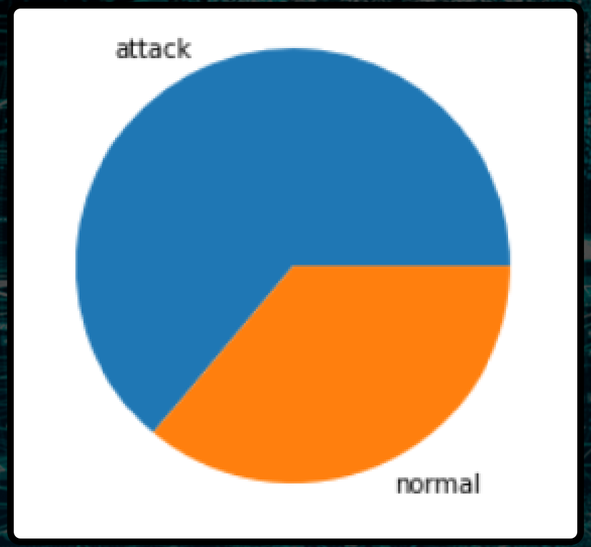
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol
6	state	nominal	Indicates to the state and its dependent proto...
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and ...
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	integer	Source to destination packet count
18	Dpkts	integer	Destination to source packet count
19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the ?ow packet size transmitted by the...

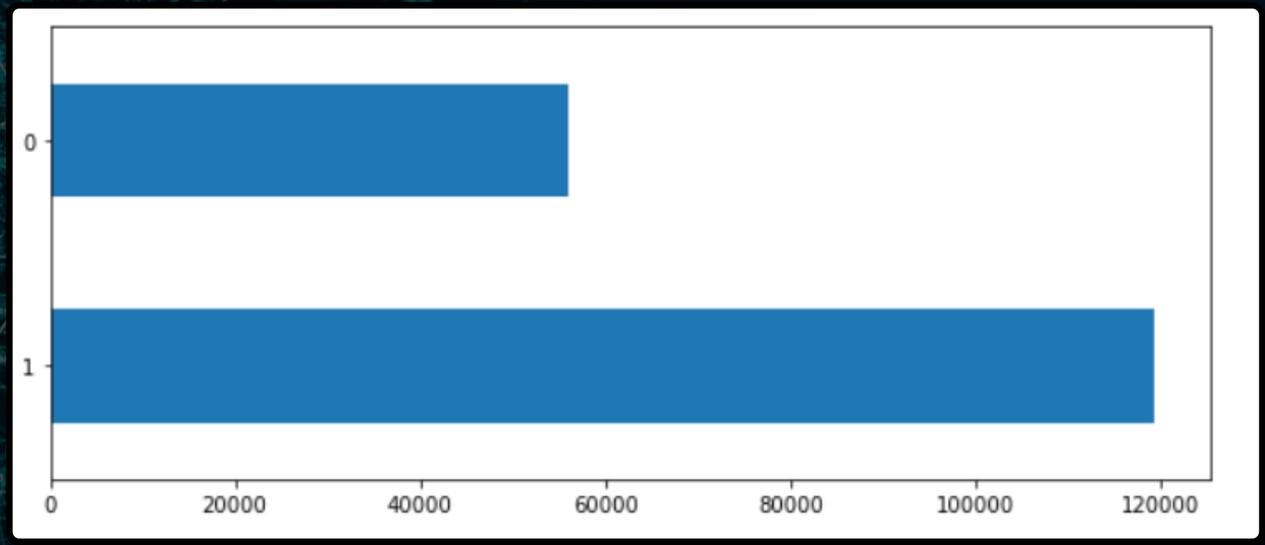
26	res_bdy_len	integer	Actual uncompressed content size of the data t...
27	Sjit	Float	Source jitter (mSec)
28	Djit	Float	Destination jitter (mSec)
29	Stime	Timestamp	record start time
30	Ltime	Timestamp	record last time
31	Sintpkt	Float	Source interpacket arrival time (mSec)
32	Dintpkt	Float	Destination interpacket arrival time (mSec)
33	tcprtt	Float	TCP connection setup round-trip time, the sum ...
34	synack	Float	TCP connection setup time, the time between th...
35	ackdat	Float	TCP connection setup time, the time between th...
36	is_sm_ips_ports	Binary	If source (1) and destination (3)IP addresses ...
37	ct_state_ttl	Integer	No. for each state (6) according to specific r...
38	ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and ...
39	is_ftp_login	Binary	If the ftp session is accessed by user and pas...
40	ct_ftp_cmd	integer	No of flows that has a command in ftp session.
41	ct_srv_src	integer	No. of connections that contain the same servi...
42	ct_srv_dst	integer	No. of connections that contain the same servi...
43	ct_dst_ltm	integer	No. of connections of the same destination add...
44	ct_src_ ltm	integer	No. of connections of the same source address ...
45	ct_src_dport_ltm	integer	No of connections of the same source address (...
46	ct_dst_sport_ltm	integer	No of connections of the same destination addr...
47	ct_dst_src_ltm	integer	No of connections of the same source (1) and t...
48	attack_cat	nominal	The name of each attack category. In this data...
49	Label	binary	0 for normal and 1 for attack records

In this first phase of exploration, we can see that there are many different types to deal with.

The dataset contains also the detailed specification of all the attributes descriptions.

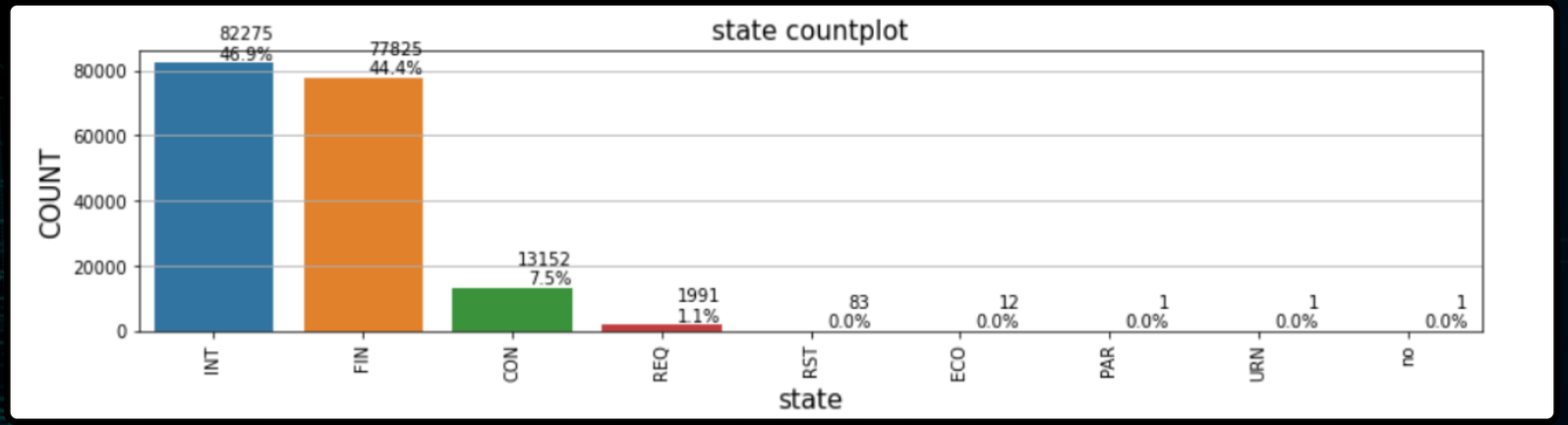
EDA



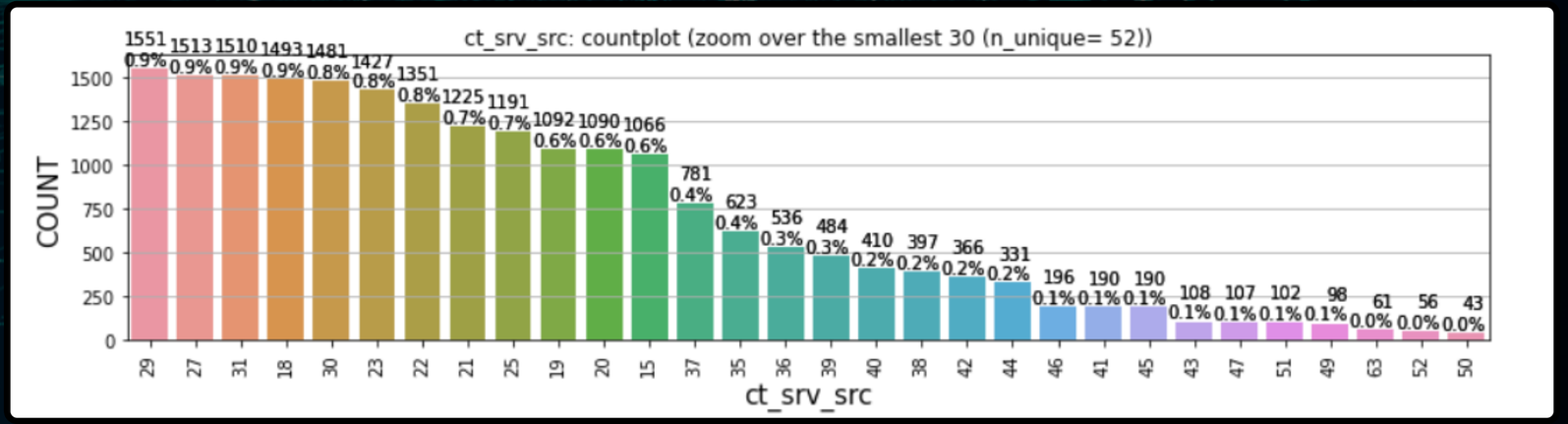


From these pictures we can spot an important aspect of the problem to solve, the class distribution is pretty balanced.

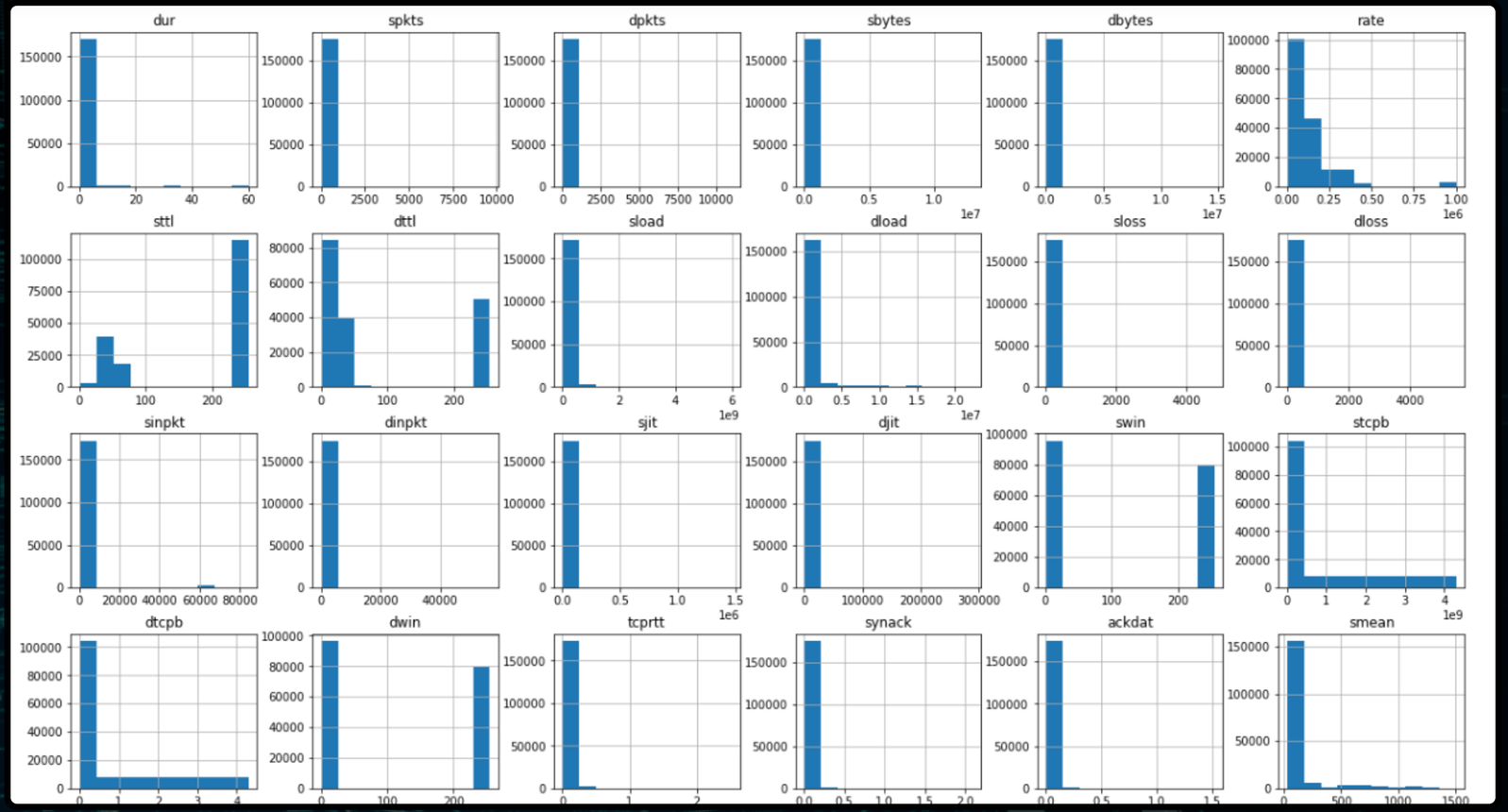
EDA



To get used to the dataset, different kinds of plots were made. Visualize data distribution is useful to deeply understand the problem.



EDA



Other insights of data distributions. In the notebook, a lot more of details can be found.

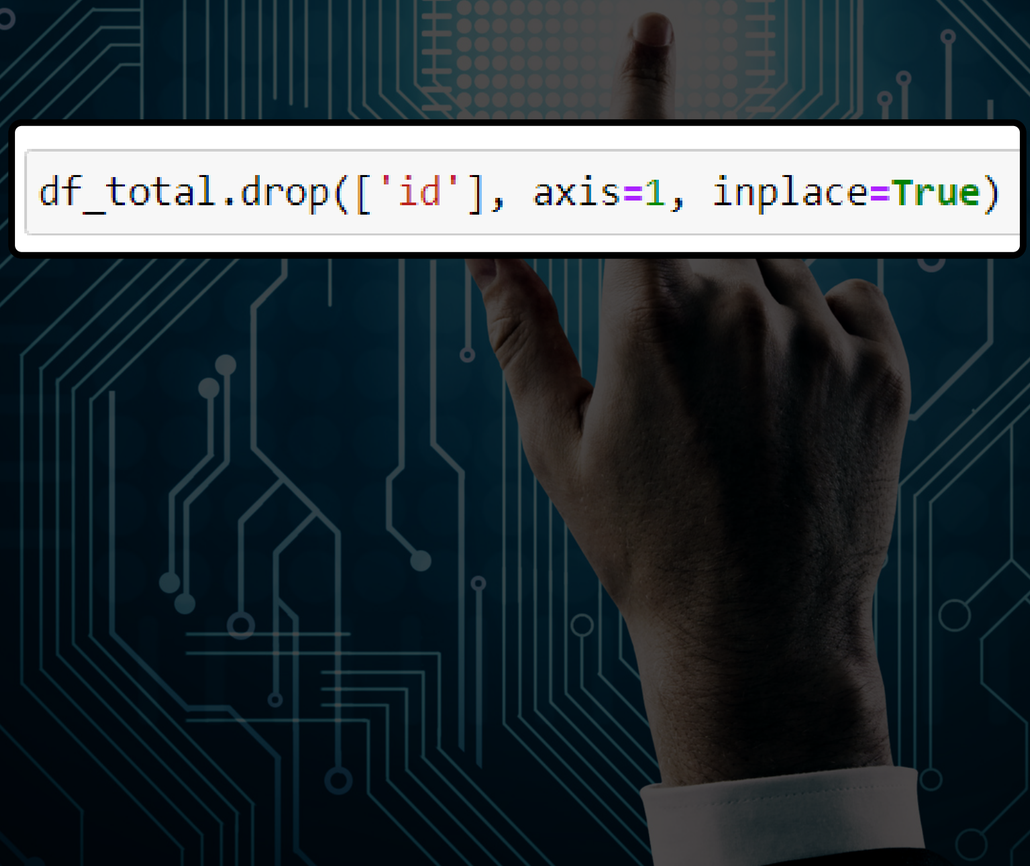
For example, some data plots were made with regard to specific attacks.

Data Cleaning

- Drop useless columns
- Missing values
- Consistency
- Single values
- Duplicates
- Balance checking

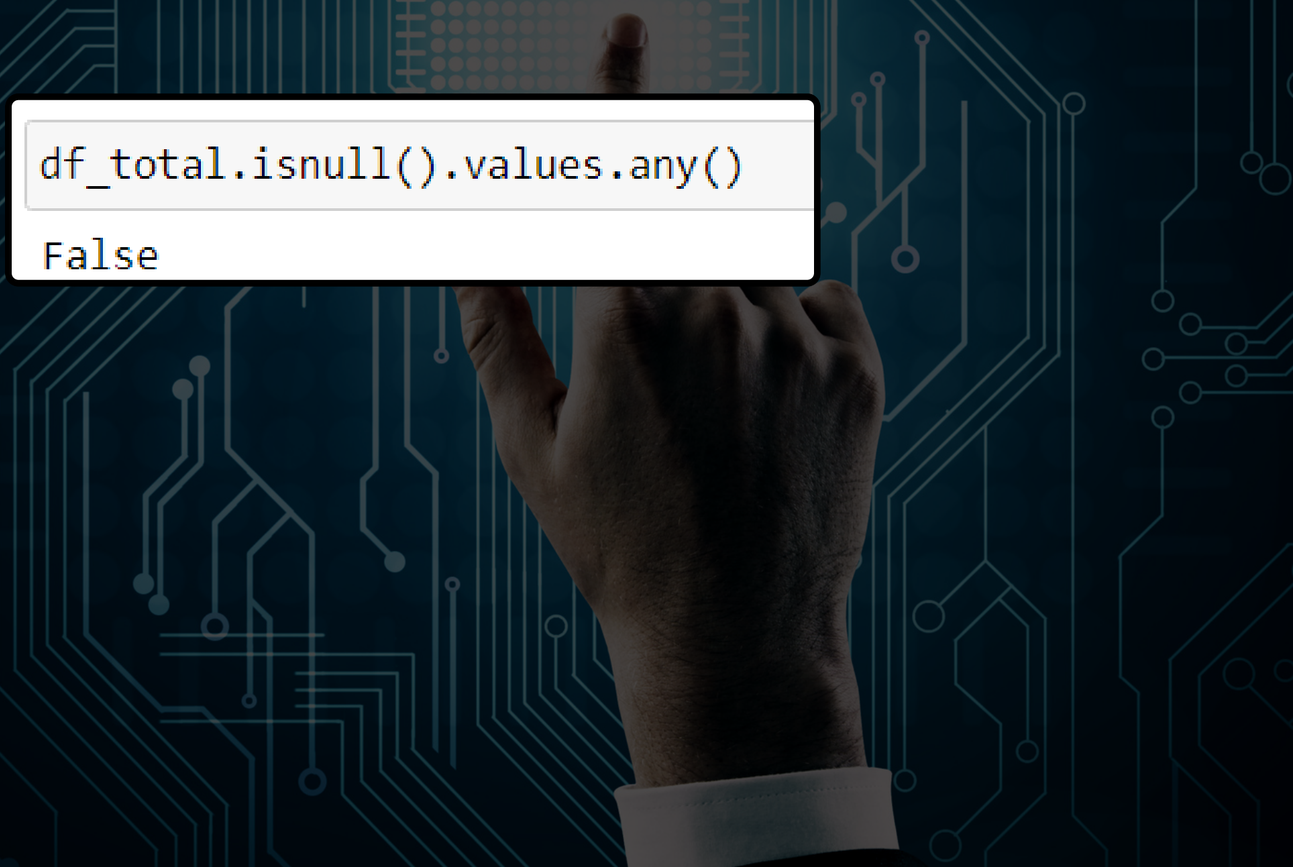
Data Cleaning

- **Drop useless columns**
- Missing values
- Consistency
- Single values
- Duplicates
- Balance checking



Data Cleaning

- Drop useless columns
- **Missing values**
- Consistency
- Single values
- Duplicates
- Balance checking



Data Cleaning

- Drop useless columns
- Missing values
- **Consistency**
- Single values
- Duplicates
- Balance checking

```
df_total[binary_cols].describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
is_ftp_login	257673.0	0.012819	0.116091	0.0	0.0	0.0	0.0	4.0
is_sm_ips_ports	257673.0	0.014274	0.118618	0.0	0.0	0.0	0.0	1.0

The `is_ftp_login` has a maximum value of 4

```
df_total.groupby(['is_ftp_login']).size()
```

```
is_ftp_login
0    254428
1      3219
2        10
4        16
dtype: int64
```

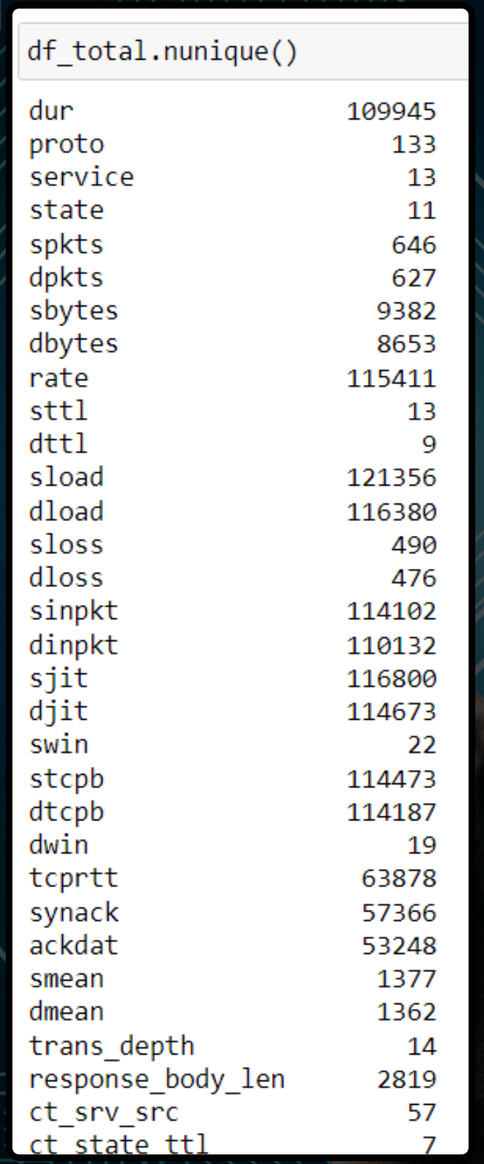
```
fig = px.histogram(train, x='is_ftp_login', log_y=True, color="attac
fig.update_layout(xaxis={'categoryorder':'total ascending'})
fig.show(renderer="svg", width=900, height=600)
```

We restore a logical value, this can also be inferred with some more precise method

```
# Let's fix this discrepancy, from the picture above we can assume:
# is_ftp_login == 2 ==> 0
# is_ftp_login == 4 ==> 1
df_total['is_ftp_login'].replace(2, 0, inplace=True)
df_total['is_ftp_login'].replace(4, 1, inplace=True)
df_total.groupby(['is_ftp_login']).size()
```

Data Cleaning

- Drop useless columns
- Missing values
- Consistency
- **Single values**
- Duplicates
- Balance checking

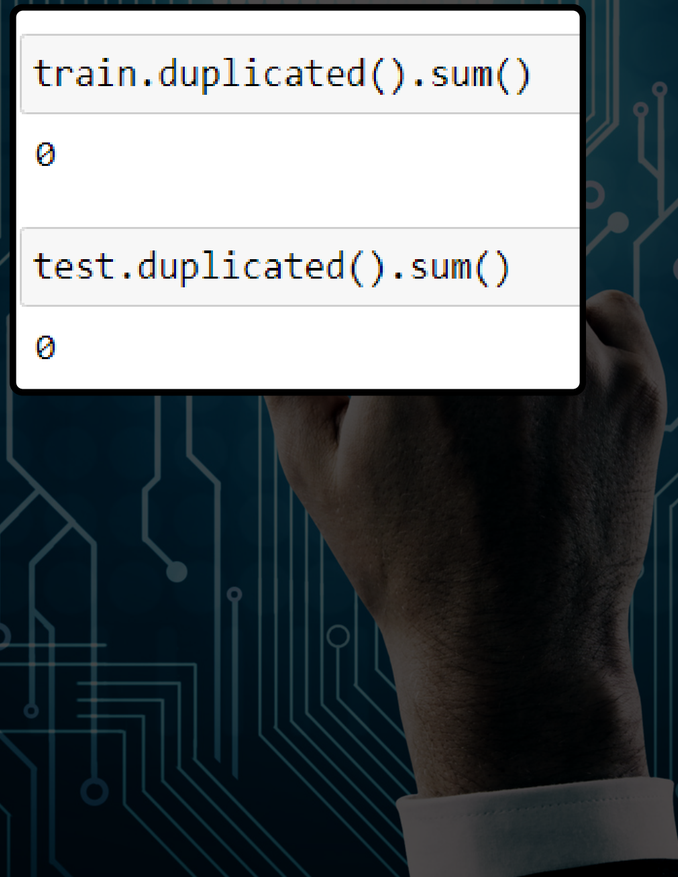


```
df_total.nunique()
```

dur	109945
proto	133
service	13
state	11
spkts	646
dpkts	627
sbytes	9382
dbytes	8653
rate	115411
sttl	13
dttl	9
sload	121356
dload	116380
sloss	490
dloss	476
sinpkt	114102
dinpkt	110132
sjit	116800
djit	114673
swin	22
stcpb	114473
dtcpb	114187
dwin	19
tcprtt	63878
synack	57366
ackdat	53248
smean	1377
dmean	1362
trans_depth	14
response_body_len	2819
ct_srv_src	57
ct_state_ttl	7

Data Cleaning

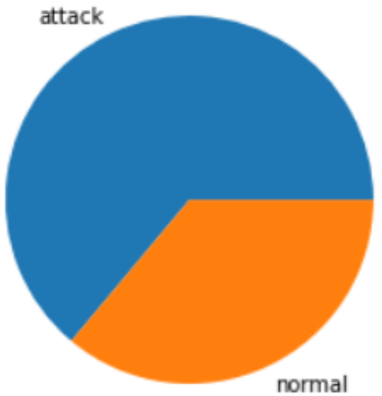
- Drop useless columns
- Missing values
- Consistency
- Single values
- **Duplicates**
- Balance checking



Data Cleaning

- Drop useless columns
- Missing values
- Consistency
- Single values
- Duplicates
- **Balance checking**

```
# binary classifiacation problem --> balanced
wedges = df_total['label'].value_counts()
plt.pie(np.array(wedges), labels= ['attack', 'normal'])
plt.show()
```



Data Preparation

State

```
col = 'state'
# create_count_df(col, train) # to set the row below
df_total.loc[~df_total[col].isin(['FIN', 'INT', 'CON', 'REQ', 'RST']), col] = 'others'
create_count_df(col)
```

	state	count	percent
0	FIN	117164	45.470034
1	INT	116438	45.188281
2	CON	20134	7.813779
3	REQ	3833	1.487544
4	RST	84	0.032599
5	others	20	0.007762

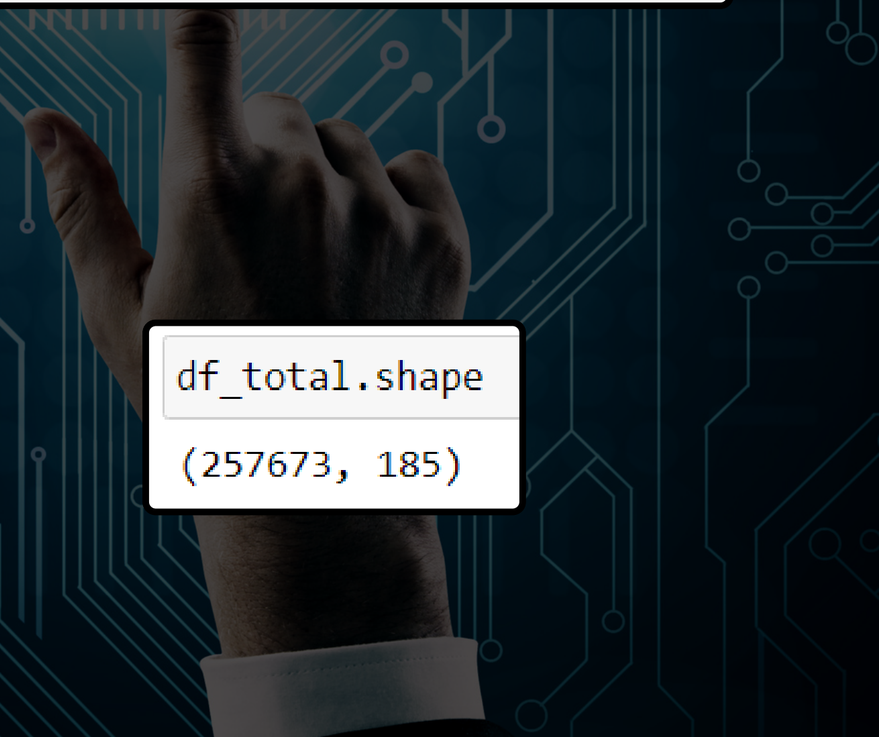
Thanks to data visualization we can conclude that there is no difference, for classification purposes, for many of the attribute values assumed by *'state'*, thus they are grouped in a single *'others'* value.
This is also extremely useful for the consequent one-hot encoding expansion, avoiding the curse of dimensionality.

Data Preparation

```
'proto_ipcomp',
'proto_aes-sp3-d',
'proto_cpnx',
'proto_srp',
'proto_sm',
'proto_mhrp',
'service_dns',
'proto_cbt',
'proto_wb-mon',
'proto_pri-enc',
'proto_ggp',
'proto_iso-tp4',
'proto_sat-expak',
'proto_wb-expak',
'proto_leaf-1',
'service_ftp-data',
'proto_br-sat-mon',
'state_INT',
```

```python
# Remove the first level of the categorical attribute, to avoid the so-called "dummy variable trap"
df_total_new = pd.get_dummies(df_total, columns=nominal_cols, drop_first=True) # one-hot encoding
dummy_variables = list(set(df_total_new) - set(df_total))
df_total = df_total_new
dummy_variables
```

One-Hot Encoding



```
df_total.shape
```

```
(257673, 185)
```

Data Preparation

Numerosity reduction and splitting

```
# retrieve original datasets
train = df_total[df_total['type'] == 'train']
train_len = len(train)
test = df_total[df_total['type'] == 'test']
test_len = len(test)

train = train.sample(n=85000, replace=True, random_state=0, ignore_index=True)
test = test.sample(n=40000, replace=True, random_state=0, ignore_index=True)

# splitting the train set
train_y_multi = train['attack_cat']
train_y_bin = train['label']
train_x_raw = train.drop(['attack_cat', 'label', 'type'], axis=1)

# splitting the test set
test_y_multi = test['attack_cat']
test_y_bin = test['label']
test_x_raw = test.drop(['attack_cat', 'label', 'type'], axis=1)
```

Correlation Analysis

```
best_features = SelectKBest(score_func=chi2, k='all')

fit_bin = best_features.fit(train_x_raw, train_y_bin)
df_scores_bin = pd.DataFrame(fit_bin.scores_)
df_cols = pd.DataFrame(train_x_raw.columns)
feature_score_bin = pd.concat([df_cols, df_scores_bin], axis=1)
feature_score_bin.columns=['feature','score']
feature_score_bin.sort_values(by=['score'], ascending=False, inplace=True)
print(feature_score_bin)
```

```
# on train data only, bin
clf_1 = RandomForestClassifier(random_state=1, n_jobs=-1)
clf_1.fit(train_x_raw, train_y_bin)
feature_importance_1 = clf_1.feature_importances_
importance_dict['train_bin'] = feature_importance_1
```

	feature	train_bin	train_multi	train_10_fold_bin	train_10_fold_multi	mean
3	sbytes	3.552020	7.329679	3.554555	7.237906	5.418540
31	ct_srv_dst	3.675137	6.891557	3.150277	5.937036	4.913502
5	sttl	9.555758	6.276070	12.228744	5.850328	8.477725
18	smean	3.051539	5.296152	2.825900	5.261388	4.108745
22	ct_srv_src	2.355446	4.458530	2.469788	4.831822	3.528897
...	...	...	...	...	...	...
172	service_radius	0.000088	0.001922	0.000154	0.002072	0.001059
140	proto_st2	0.002719	0.002246	0.001637	0.001561	0.002041
181	state_others	0.002595	0.001020	0.002367	0.000785	0.001692
69	proto_icmp	0.003395	0.000660	0.002001	0.000741	0.001699
123	proto_rtp	0.000766	0.000198	0.000679	0.000176	0.000455

182 rows × 6 columns

```
importance_df = importance_df.sort_values('mean', ascending=False)
```

Feature Selection

```
irrelevant_features_chi2 = list(feature_score_bin['feature'].iloc[30:])
irrelevant_features_rf = list(importance_df['feature'].iloc[30:])
irrelevant_features = list(set(irrelevant_features_rf).union(set(irrelevant_features_chi2)))
irrelevant_features
```

```
['smean',
 'proto_nvp',
 'proto_trunk-1',
 'proto_irtp',
 'state_FIN',
 'response_body_len',
 'proto_leaf-2',
 'proto_rtp',
 'proto_secure-vmtp',
 'service_http',
 'proto_argus',
 'proto_ib',
 'proto_wsn',
 'proto_visa',
 'proto_ipcv',
 'proto_vmtp',
 'tcprtt',
 'proto_idpr-cmtp',
 'proto_egp',
```

```
train_reduced = train_x_raw.drop(labels=irrelevant_features, axis=1)
test_reduced = test_x_raw.drop(labels=irrelevant_features, axis=1)
train_reduced.shape
```

```
(85000, 24)
```

Feature Selection

```
correlations_reduced = train_reduced.corr(method='pearson') # Pearson is needed for numerical data
top_correlations(correlations_reduced)
```

```
rate <-> sload  ==  97.26444887237412
dload <-> dmean  ==  96.568000517313
stcpb <-> dtcpb  ==  99.62538761109137
ct_srv_src <-> ct_dst_src_ltm  ==  96.6588291444802
ct_srv_src <-> ct_srv_dst  ==  97.94474336742273
ct_dst_ltm <-> ct_src_dport_ltm  ==  95.27340882047028
ct_dst_src_ltm <-> ct_srv_dst  ==  97.15323413610867
```

```
selected_but_correlated = ['rate', 'dmean', 'dtcpb', 'ct_dst_src_ltm', 'ct_srv_dst', 'ct_src_dport_ltm']
train_reduced = train_reduced.drop(labels=selected_but_correlated, axis=1)
test_reduced = test_reduced.drop(labels=selected_but_correlated, axis=1)
train_reduced.shape
```

```
(85000, 18)
```

Normalization

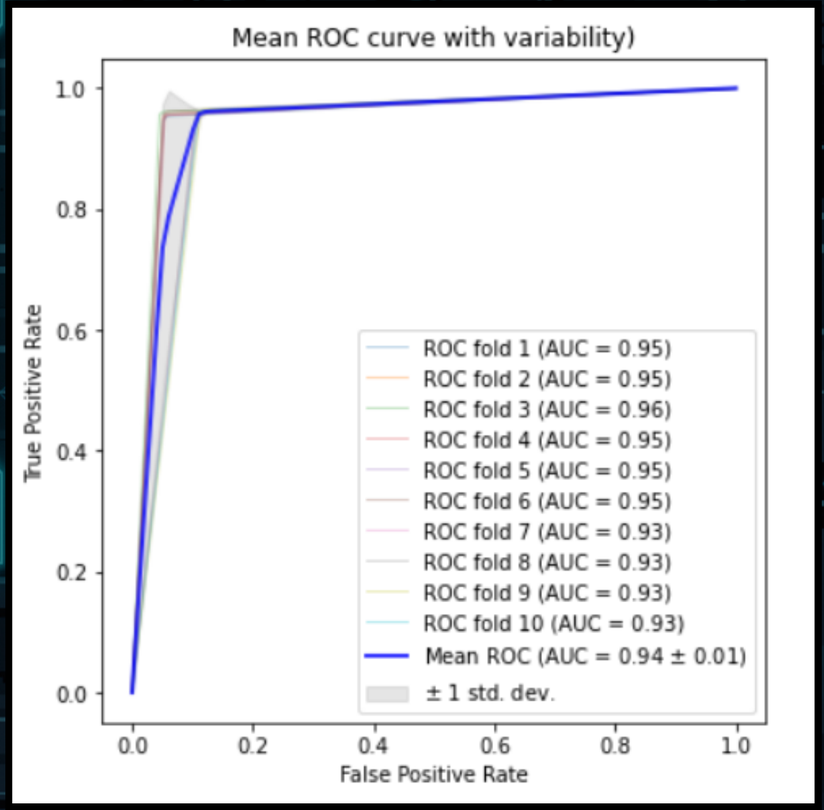
```
from sklearn.preprocessing import MinMaxScaler
numeric_cols = train_reduced.select_dtypes(exclude='object').columns
mms = MinMaxScaler()
train_reduced[numeric_cols] = mms.fit_transform(train_reduced[numeric_cols])
test_reduced[numeric_cols] = mms.transform(test_reduced[numeric_cols])
```

```
train_reduced.describe()
```

	dur	spkts	dpkts	sbytes	sttl	dttl	sload
count	8.500000e+04	85000.000000	85000.000000	85000.000000	85000.000000	85000.000000	85000.000000
mean	2.267015e-02	0.002024	0.001710	0.000697	0.703783	0.318392	0.013064
std	1.080474e-01	0.014814	0.009499	0.014115	0.403821	0.439560	0.034747
min	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.333334e-07	0.000104	0.000000	0.000005	0.243137	0.000000	0.000002
50%	2.875001e-05	0.000312	0.000184	0.000034	0.996078	0.115079	0.000148
75%	1.120560e-02	0.001144	0.000922	0.000106	0.996078	1.000000	0.015873
max	1.000000e+00	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Classification
Decision Tree

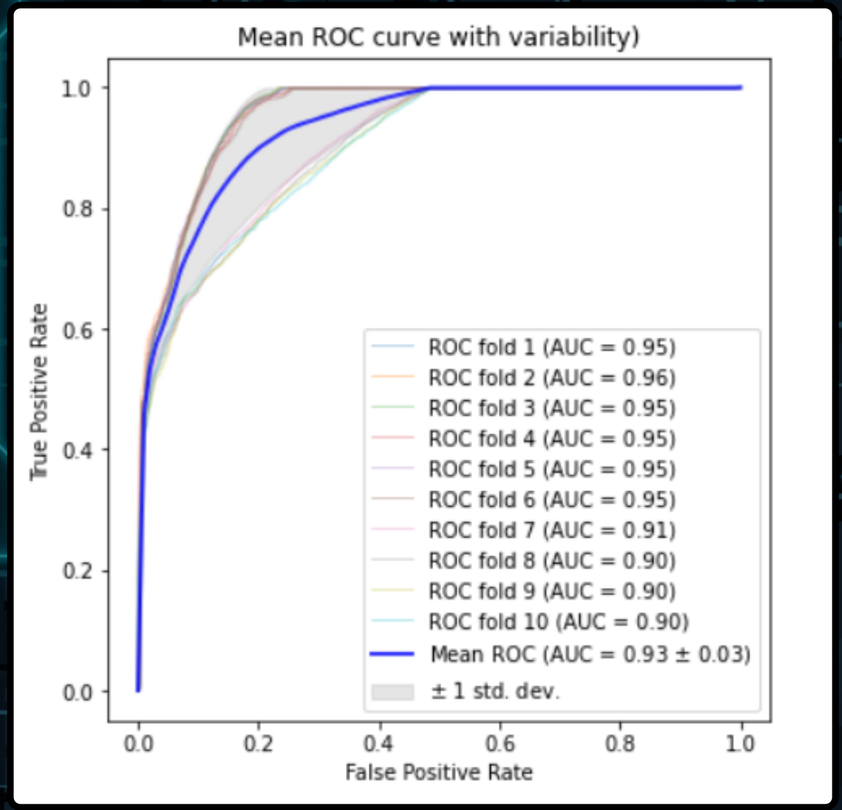
- parameters setting
- decision tree visualization
- ccp_alpha testing for pruning
- overfitting testing
- confusion matrix
- cross-validated (k=10)



Average F1-score == 94.04

Classification
Logistic Regression

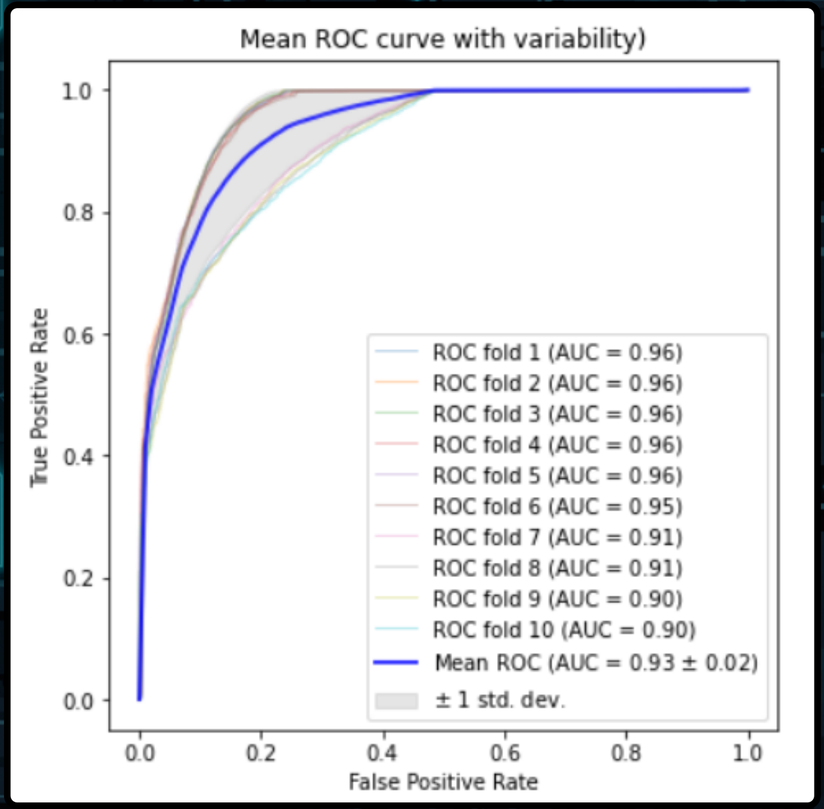
- parameters setting:
  - dual=False
  - C=0.1
  - class_weight=balanced
  - solver=lbfgs
  - maax_iter=10.000
- cross-validated (k=10)



Average F1-score == 85.93

Classification
Support Vector Machine

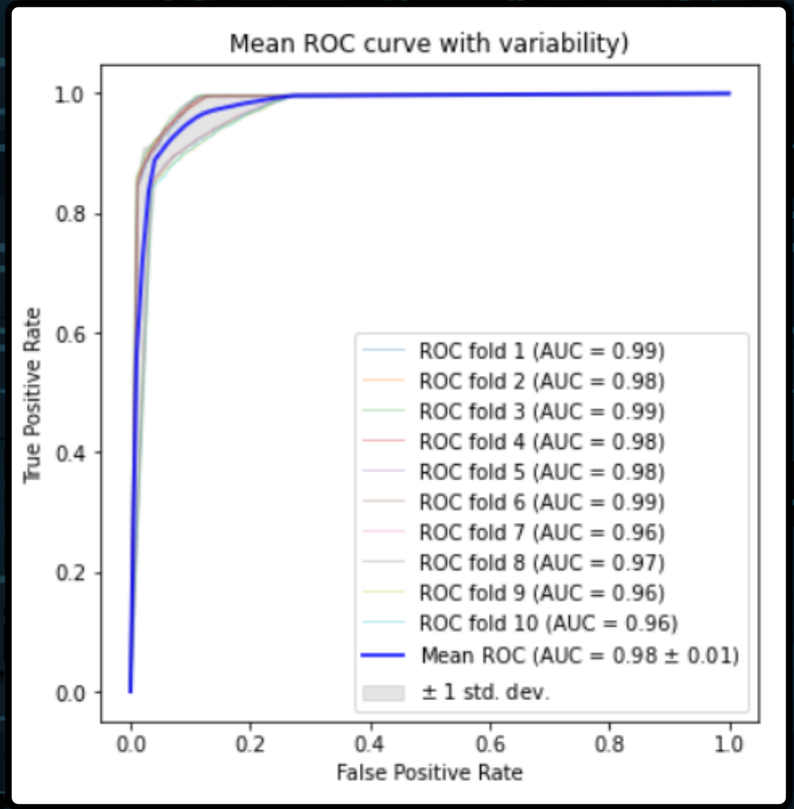
- parameters setting:
  - duale=False
  - tol=1e-05
  - class_weight=balanced
  - max_iter=10.000
- LinearSVC
- cross-validated (k=10)



Average F1-score == 86.06

Classification
K-Nearest Neighbours

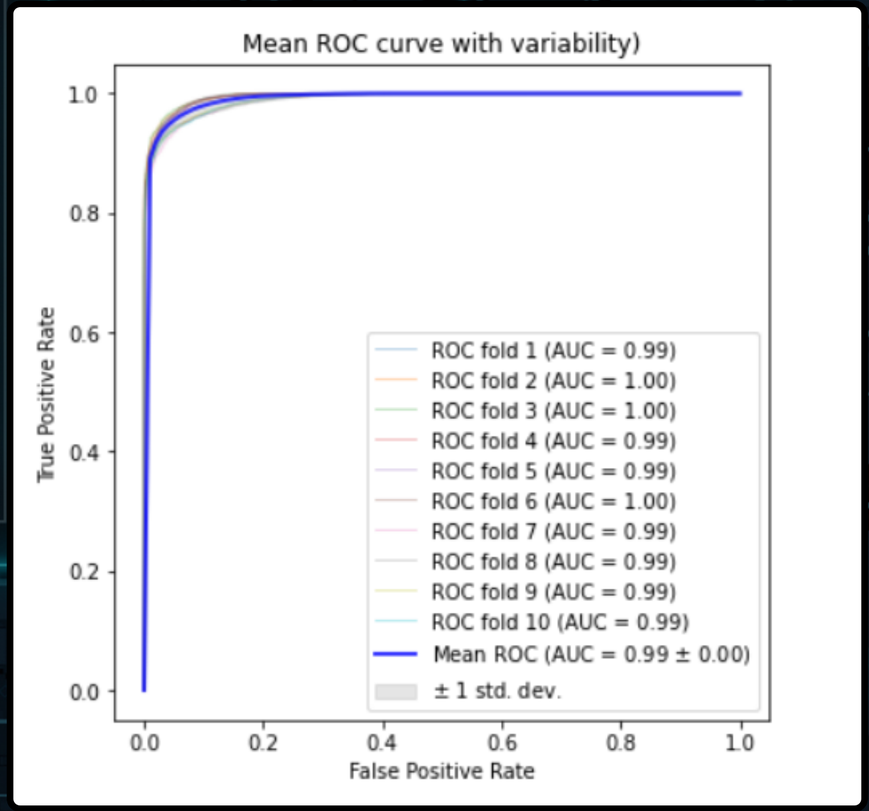
- parameters setting:
  - n_neighbors=7
  - weights=distance
  - algorithm=auto
- corss-validated (k=10)



Average F1-score == 92.48

Classification
Random Forest

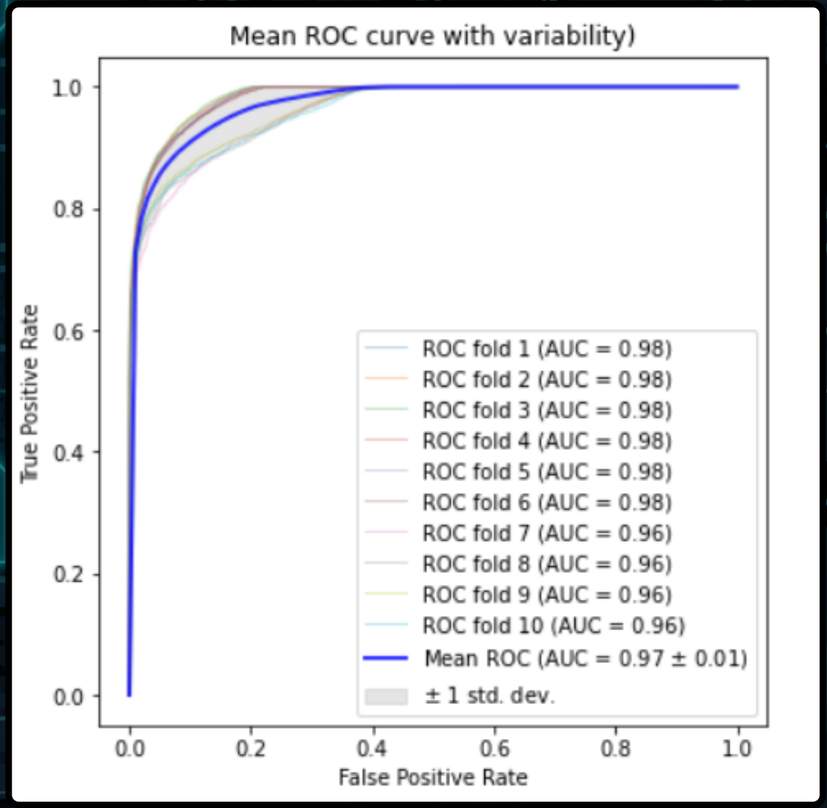
- parameter setting:
  - class_weight=balanced
- cross-validated (k=10)



Average F1-score == 94.89

Classification
Multi-Layer Perceptron

- parameter setting:
  - solver=adam
  - max_iterations=10.000
- cross-validated (k=10)



Average F1-score == 89.50

Validation

paired_ttest_5x2cv test with 9 degrees of freedom and alpha = 0.05

	Decision Tree	Logistic Regression	SVM	KNN	Random Forest	MLP
Decision Tree						
Logistic Regression	p=0 t=17.553					
SVM	p=0 t=14.641	p=0.973 t=-0.036				
KNN	p=0.005 t=4.834	p=0 t=-14.162	p=0 t=-22.209			
Random Forest	p=0 t=-10.389	p=0 t=-58.135	p=0 t=-61.495	p=0 t=-11.747		
MLP	p=0 t=30.326	p=0 t=-14.514	p=0 t=-17.199	p=0 t=9.989	p=0 t=13.049	

Future Work and Conclusions

- Improve the data cleaning process and approach some feature engineering
- Experimenting a lot more with the parameters of the chosen classifiers
- Evaluate the performance of new advanced classifiers
- Run the project with the full dataset
- Evaluate clustering tendency and clustering techniques
- Approach the problem as an outlier detection problem
- Complete the analysis of the multi-class classification problem

Thank You!

Francesco Venturini