initial_tests

March 31, 2024

1 Introduction

Point of this notebook is simply to observe (withotu any decision making) the performance across a matrix of window skips and window lengths

[ ]:
```
from sampleddetection.environment.datastructures import Action, State
from sampleddetection.environment.model import Environment
from sampleddetection.datastructures.flowsession import SampledFlowSession
import numpy as np
from typing import List
import os
from tqdm.notebook import tqdm
from pathlib import Path
from itertools import product
import random

# Make sure these are reloaded when cells are rerun
%load_ext autoreload
%autoreload 2
```

[ ]:
```
# Setup the environment
# From Microsecond to dekasecond
window_skips    = np.logspace(-6, 1, 4, dtype=float)
window_lengths  = np.logspace(-6, -3, 4, dtype=float)
#window_lengths  = 2*np.linspace(0.01, , 3, dtype=float)
batch_size      = 16
csv_path = './data/missingHeartbleedWed.csv'
dataset_dir    = './data/precalc_windows/'
dataset_filename = 'ws_{}_wl_{}.csv'
desired_features = [
            # Debugging info
            "start_ts",
            "start_timestamp",
            "end_timestamp",
            "tot_fwd_pkts",
            "tot_bwd_pkts",
            # Non debugging
            "label",
```

```
            "fwd_pkt_len_max",
            "fwd_pkt_len_min",
            "fwd_pkt_len_mean",
            "bwd_pkt_len_max",
            "bwd_pkt_len_min",
            "bwd_pkt_len_mean",
            "flow_byts_s",
            "flow_pkts_s",
            "flow_iat_mean",
            "flow_iat_max",
            "flow_iat_min",
            "fwd_iat_mean",
            "fwd_iat_max",
            "fwd_iat_min",
            "bwd_iat_max",
            "bwd_iat_min",
            "bwd_iat_mean",
            "pkt_len_min",
            "pkt_len_max",
            "pkt_len_mean",
]

# Use product to get a matrix of combinations
options_matrix = list(product(window_skips, window_lengths))
print(f"Working with {len(options_matrix)} permutaitions")
```

```
Working with 16 permutaitions
```

[ ]:

```
# Create or Load dataset
from sampleddetection.samplers.window_sampler import DynamicWindowSampler
from sampleddetection.writers.convenience import save_flows_to_csv
from sampleddetection.readers.readers import CSVReader
from sampleddetection.common_lingo import Attack, ATTACK_TO_STRING

sampler = DynamicWindowSampler(Path(csv_path))
environment = Environment(sampler)
min_necessary_classes = batch_size * 128

samples_per_class = {Attack.BENIGN: min_necessary_classes, Attack.GENERAL:
 ↪min_necessary_classes}

# Create it
    # Ensure that the dataset is balanced.
```

```
2024-03-31 20:30:06,824 - DynamicWindowSampler - INFO - Loading the capture file
data/missingHeartbleedWed.csv
2024-03-31 20:30:06,824 - DynamicWindowSampler - INFO - Loading the capture file
data/missingHeartbleedWed.csv
```

```
2024-03-31 20:30:06,826 - CSVReader - INFO - Reading csv…
2024-03-31 20:30:06,824 - DynamicWindowSampler - INFO - Loading the capture file
data/missingHeartbleedWed.csv
2024-03-31 20:30:06,826 - CSVReader - INFO - Reading csv…
2024-03-31 20:30:37,284 - CSVReader - INFO - CSV loaded, took  30.46 seconds
with 13704955 length
```

[ ]:
```python
from sampleddetection.datastructures.flow import Flow
from typing import Dict, Tuple

def generate_sessions(amount: int, ws: float, wl: float) ->␣
 ↪List[Tuple[Tuple,Flow]]:
    """Ensure we get a balanced sampling from the large dataset."""
    cur_amnt = 0
    flows: List[Tuple,Flow] = []
    inner_bar = tqdm(total=min_necessary_classes*2,desc=f'Generating ws: {ws}-␣
 ↪wl: {wl} flow',leave=False)
    count_per_class = {Attack.BENIGN: 0, Attack.GENERAL: 0}
    distributions = {Attack.BENIGN: 0, Attack.GENERAL: 0}

    while sum(list(count_per_class.values())) < amount*2:

        flow_sesh =  environment.reset(winskip=ws,winlen=wl).flow_sesh
        #amnt_sesh_flows = len(flow_sesh.flows.keys())
        # Count the distributions
        label_distributions = flow_sesh.flow_label_distribution()
        # For now just predict binary attack-benining
        for kflow, flow in flow_sesh.flows.items() :
            label = Attack.GENERAL if flow.label != Attack.BENIGN else Attack.
 ↪BENIGN

            if count_per_class[label] >= min_necessary_classes:
                continue # Dont over add

            count_per_class[label] += 1

            flows.append((kflow,flow))

            inner_bar.update(1)
    return flows
```

[ ]:
```python
flows = {}
# Set random seeds:
np.random.seed(0)
random.seed(0)
import csv
```

```python
# Generate the datasets
for ws, wl in tqdm(options_matrix,desc='Creating datasets'):
    # Check if datasets exists
    flows = {f"ws:{ws}-ws:{wl}" : []}
    target_name = os.path.join(dataset_dir,dataset_filename.format(ws, wl))
    if os.path.exists(target_name):
        print(f"Will later be Loading {dataset_filename.format(ws, wl)} from␣
↪{dataset_dir}")
        continue
    sesh = generate_sessions(min_necessary_classes,ws,wl)

    ds_path = os.path.join(dataset_dir,dataset_filename.format(ws, wl))
    save_flows_to_csv(sesh, ds_path, desired_features=desired_features,␣
↪samples_per_class=samples_per_class,overwrite=True)
```

```
Creating datasets:   0%|          | 0/16 [00:00<?, ?it/s]

Will later be Loading ws_1e-06_wl_1e-06.csv from ./data/precalc_windows/
Will later be Loading ws_1e-06_wl_1e-05.csv from ./data/precalc_windows/
Will later be Loading ws_1e-06_wl_0.0001.csv from ./data/precalc_windows/
Will later be Loading ws_1e-06_wl_0.001.csv from ./data/precalc_windows/
Will later be Loading ws_0.00021544346900318845_wl_1e-06.csv from
./data/precalc_windows/
Will later be Loading ws_0.00021544346900318845_wl_1e-05.csv from
./data/precalc_windows/
Will later be Loading ws_0.00021544346900318845_wl_0.0001.csv from
./data/precalc_windows/
Will later be Loading ws_0.00021544346900318845_wl_0.001.csv from
./data/precalc_windows/
Will later be Loading ws_0.046415888336127826_wl_1e-06.csv from
./data/precalc_windows/
Will later be Loading ws_0.046415888336127826_wl_1e-05.csv from
./data/precalc_windows/
Will later be Loading ws_0.046415888336127826_wl_0.0001.csv from
./data/precalc_windows/
Will later be Loading ws_0.046415888336127826_wl_0.001.csv from
./data/precalc_windows/
Will later be Loading ws_10.0_wl_1e-06.csv from ./data/precalc_windows/
Will later be Loading ws_10.0_wl_1e-05.csv from ./data/precalc_windows/
Will later be Loading ws_10.0_wl_0.0001.csv from ./data/precalc_windows/
Will later be Loading ws_10.0_wl_0.001.csv from ./data/precalc_windows/
```

[ ]:
```python
# Load PreCalced datasets
for ws, wl in tqdm(options_matrix,desc='Loading datasets'):
    target_name = os.path.join(dataset_dir,dataset_filename.format(ws, wl))
    if not os.path.exists(target_name):
```

```
        print(f"Could not find {target_name}")
        raise FileNotFoundError
```

```
Loading datasets:   0%|          | 0/16 [00:00<?, ?it/s]
```

2 Training Model on Different Schedules

We will use the matrix of different parameters to see how the training changes performance.

[ ]:
```
# This will be a function that will take flows calculated/loaded up above and␣
 ↪will train the model.
# It will return data of  the training and testing results to later be plotted␣
 ↪in a loop that will call it
import pandas as pd
from sampleddetection.util.data import clean_dataset,train_classifier_XGBoost
from sklearn.model_selection import train_test_split
features = [
            "label",
            "fwd_pkt_len_max",
            "fwd_pkt_len_min",
            "fwd_pkt_len_mean",
            "bwd_pkt_len_max",
            "bwd_pkt_len_min",
            "bwd_pkt_len_mean",
            "flow_byts_s",
            "flow_pkts_s",
            "flow_iat_mean",
            "flow_iat_max",
            "flow_iat_min",
            "fwd_iat_mean",
            "fwd_iat_max",
            "fwd_iat_min",
            "bwd_iat_max",
            "bwd_iat_min",
            "bwd_iat_mean",
            "pkt_len_min",
            "pkt_len_max",
            "pkt_len_mean",
]
attacks_to_detect = [
    Attack.SLOWLORIS,
    Attack.SLOWHTTPTEST,
    Attack.HULK,
    Attack.GOLDENEYE,
    Attack.HEARTBLEED
]
```

```python
def evaluate_performance(df: pd.DataFrame, ws: float, wl: float) -> Dict:
    """Evaluate the performance of the model with the given dataset."""
    # Clean the dataset
    df_ddos = clean_dataset(df,features, attacks_to_detect)

    # Train the Model
    X_train, X_test, y_train, y_test = train_test_split(
        df_ddos.drop(columns=["label"]), df_ddos["label"], test_size=0.3
    )

    mode, evals = train_classifier_XGBoost(X_train,  y_train,X_test, y_test)

    return evals
```

[ ]:

```python
# This will be the outer loop that will vall evaluete_performance
accuracies = []
log_losses = []
roc_aucs = []
for ws, wl in tqdm(options_matrix,desc='Evaluating datasets'):
    target_name = os.path.join(dataset_dir,dataset_filename.format(ws, wl))
    if not os.path.exists(target_name):
        print(f"Could not find {target_name}")
        raise FileNotFoundError
    # Load the data
    df = pd.read_csv(target_name)
    # Evaluate the data
    metrics = evaluate_performance(df, ws, wl)
    accuracies.append(metrics["accuracy"])
    log_losses.append(metrics["log_loss"])
    roc_aucs.append(metrics["roc_auc"])

# Plot
```

```
Evaluating datasets:   0%|          | 0/16 [00:00<?, ?it/s]

label
1    1200
0    1200
 44%|        | 11/25 [00:03<00:04,  3.18trial/s, best loss:
-0.8869047619047619]
Best parameters: {'gamma': 0, 'max_depth': 35, 'min_child_weight': 4.0,
'n_estimators': 20, 'subsample': 0.8}
label
1    1200
0    1200
Name: count, dtype: int64
 52%|       | 13/25 [00:01<00:00, 12.61trial/s, best loss:
```

```
-0.9077380952380952]
Best parameters: {'gamma': 0, 'max_depth': 45, 'min_child_weight': 5.0,
'n_estimators': 70, 'subsample': 0.9}
label
1    1200
0    1200
Name: count, dtype: int64
 60%|       | 15/25 [00:01<00:00, 13.31trial/s, best loss:
-0.8934523809523809]
Best parameters: {'gamma': 0, 'max_depth': 15, 'min_child_weight': 1.0,
'n_estimators': 70, 'subsample': 0.9}
label
1    1200
0    1200
Name: count, dtype: int64
 36%|        | 9/25 [00:00<00:01, 13.26trial/s, best loss:
-0.9130952380952382]
Best parameters: {'gamma': 0, 'max_depth': 15, 'min_child_weight': 6.0,
'n_estimators': 50, 'subsample': 1}
label
1    1200
0    1200
Name: count, dtype: int64
 28%|        | 7/25 [00:00<00:01, 12.30trial/s, best loss: -0.893452380952381]
Best parameters: {'gamma': 1, 'max_depth': 25, 'min_child_weight': 2.0,
'n_estimators': 90, 'subsample': 1}
label
0    1200
1    1200
Name: count, dtype: int64
 40%|        | 10/25 [00:00<00:01, 13.14trial/s, best loss:
-0.9107142857142858]
Best parameters: {'gamma': 0, 'max_depth': 20, 'min_child_weight': 2.0,
'n_estimators': 30, 'subsample': 1}
label
0    1200
1    1200
Name: count, dtype: int64
 40%|        | 10/25 [00:00<00:01, 12.25trial/s, best loss:
-0.913095238095238]
Best parameters: {'gamma': 2, 'max_depth': 5, 'min_child_weight': 6.0,
'n_estimators': 60, 'subsample': 1}
label
0    1200
1    1200
Name: count, dtype: int64
 36%|        | 9/25 [00:00<00:01, 11.90trial/s, best loss:
-0.8994047619047618]
```

```
Best parameters: {'gamma': 0, 'max_depth': 30, 'min_child_weight': 4.0,
'n_estimators': 80, 'subsample': 0.9}
label
1    1200
0    1200
Name: count, dtype: int64
 20%|          | 5/25 [00:00<00:01, 11.57trial/s, best loss: -0.880952380952381]
Best parameters: {'gamma': 0, 'max_depth': 45, 'min_child_weight': 2.0,
'n_estimators': 50, 'subsample': 0.9}
label
1    1200
0    1200
Name: count, dtype: int64
 36%|          | 9/25 [00:00<00:01, 13.43trial/s, best loss: -0.893452380952381]
Best parameters: {'gamma': 1, 'max_depth': 50, 'min_child_weight': 6.0,
'n_estimators': 100, 'subsample': 1}
label
0    1200
1    1200
Name: count, dtype: int64
 52%|          | 13/25 [00:00<00:00, 13.37trial/s, best loss:
-0.8845238095238095]
Best parameters: {'gamma': 1, 'max_depth': 10, 'min_child_weight': 6.0,
'n_estimators': 90, 'subsample': 0.7}
label
1    1200
0    1200
Name: count, dtype: int64
 36%|          | 9/25 [00:00<00:01, 10.88trial/s, best loss:
-0.9071428571428571]
Best parameters: {'gamma': 0, 'max_depth': 45, 'min_child_weight': 9.0,
'n_estimators': 80, 'subsample': 0.8}
label
0    1200
1    1200
Name: count, dtype: int64
 52%|          | 13/25 [00:00<00:00, 14.60trial/s, best loss:
-0.8827380952380952]
Best parameters: {'gamma': 0, 'max_depth': 45, 'min_child_weight': 9.0,
'n_estimators': 90, 'subsample': 0.8}
label
0    1200
1    1200
Name: count, dtype: int64
 32%|          | 8/25 [00:00<00:01, 13.44trial/s, best loss: -0.9]
Best parameters: {'gamma': 0, 'max_depth': 20, 'min_child_weight': 5.0,
'n_estimators': 90, 'subsample': 0.8}
label
```

```
1    1200
0    1200
Name: count, dtype: int64
 28%|          | 7/25 [00:00<00:01, 12.12trial/s, best loss: -0.8875]
Best parameters: {'gamma': 1, 'max_depth': 15, 'min_child_weight': 5.0,
'n_estimators': 50, 'subsample': 0.8}
label
1    1200
0    1200
Name: count, dtype: int64
 28%|          | 7/25 [00:00<00:01, 13.26trial/s, best loss:
-0.8886904761904763]
Best parameters: {'gamma': 0, 'max_depth': 10, 'min_child_weight': 1.0,
'n_estimators': 20, 'subsample': 0.8}
```

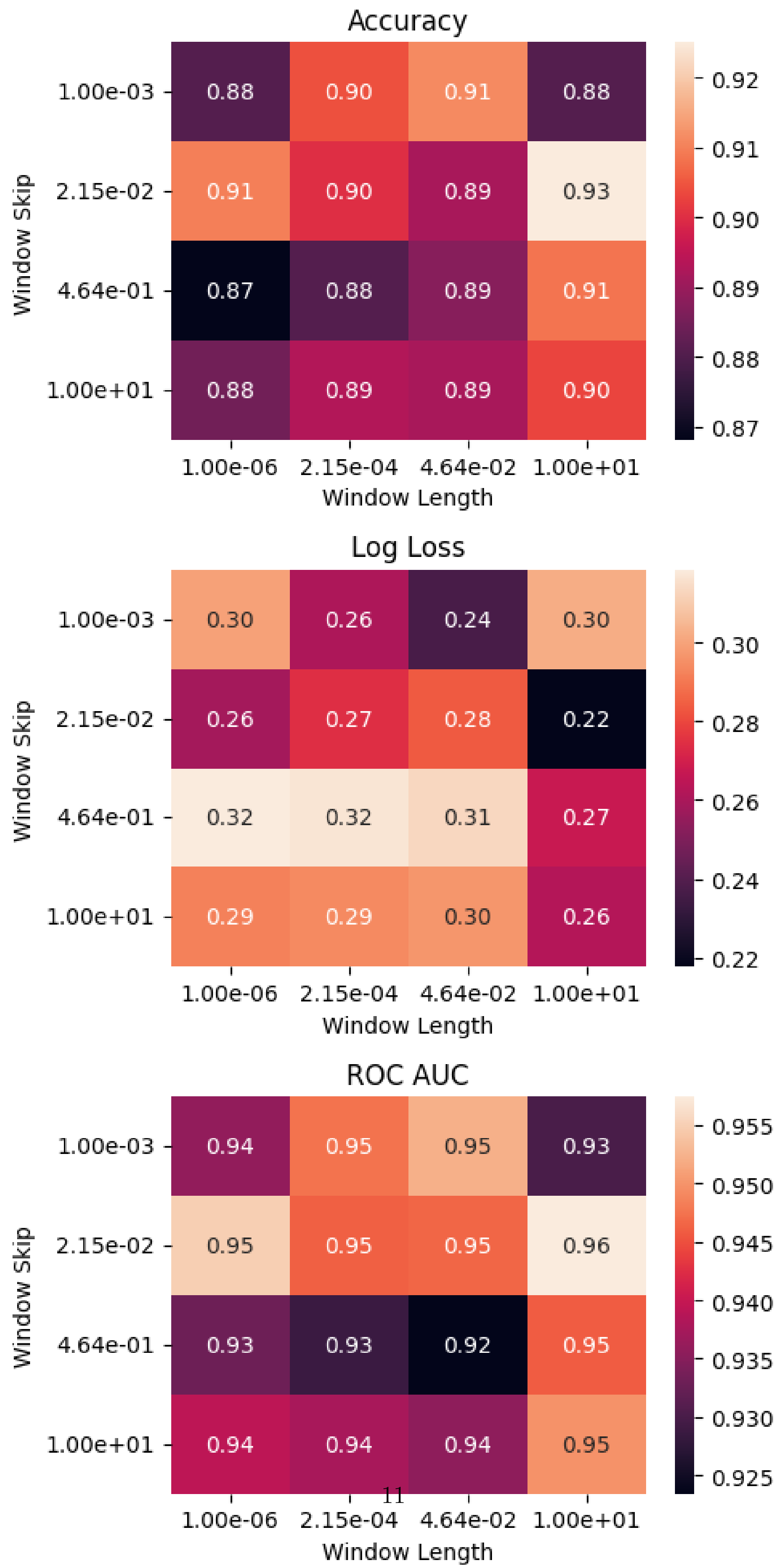
[ ]:
```
print(options_matrix)
```

```
[(1e-06, 1e-06), (1e-06, 1e-05), (1e-06, 0.0001), (1e-06, 0.001),
(0.00021544346900318845, 1e-06), (0.00021544346900318845, 1e-05),
(0.00021544346900318845, 0.0001), (0.00021544346900318845, 0.001),
(0.04641588833612782, 1e-06), (0.04641588833612782, 1e-05),
(0.04641588833612782, 0.0001), (0.04641588833612782, 0.001), (10.0, 1e-06),
(10.0, 1e-05), (10.0, 0.0001), (10.0, 0.001)]
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import LogFormatter
from matplotlib.ticker import FormatStrFormatter

#winskips = [f"{ws:.2f}" for ws, wl in options_matrix]
#winlens = [f"{wl:.2f}" for ws, wl in options_matrix]
winskips = [f"{i:.2e}" for i in np.logspace(-6, 1, 4)]
winlens = [f"{i:.2e}" for i in np.logspace(-3, 1, 4)]
# Format these with scientific (1e-6) notation

fig, ax = plt.subplots(3,1,figsize=(5,10))

#formatter = LogFormatter(10, labelOnlyBase=False)

print(winskips)

sns.heatmap(np.array(accuracies).reshape(4,4),ax=ax[0],annot=True,fmt=".2f",
 xticklabels=winskips, yticklabels=winlens)
ax[0].set_title("Accuracy")
ax[0].set_xlabel("Window Length")
ax[0].set_ylabel("Window Skip")
```

```
sns.heatmap(np.array(log_losses).reshape(4,4),ax=ax[1],annot=True,fmt=".
↪2f",xticklabels=winskips, yticklabels=winlens)
ax[1].set_title("Log Loss")
ax[1].set_xlabel("Window Length")
ax[1].set_ylabel("Window Skip")

sns.heatmap(np.array(roc_aucs).reshape(4,4),ax=ax[2],annot=True,fmt=".
↪2f",xticklabels=winskips, yticklabels=winlens)
ax[2].set_title("ROC AUC")
ax[2].set_xlabel("Window Length")
ax[2].set_ylabel("Window Skip")

plt.tight_layout()
plt.show()
```

```
['1.00e-06', '2.15e-04', '4.64e-02', '1.00e+01']
```


Accuracy
Window Skip
Window Length
1.00e-03
2.15e-02
4.64e-01
1.00e+01
1.00e-06
2.15e-04
4.64e-02
1.00e+01
0.88 0.90 0.91 0.88
0.91 0.90 0.89 0.93
0.87 0.88 0.89 0.91
0.88 0.89 0.89 0.90
0.92
0.91
0.90
0.89
0.88
0.87
Log Loss
Window Skip
Window Length
0.30 0.26 0.24 0.30
0.26 0.27 0.28 0.22
0.32 0.32 0.31 0.27
0.29 0.29 0.30 0.26
0.30
0.28
0.26
0.24
0.22
ROC AUC
Window Skip
Window Length
0.94 0.95 0.95 0.93
0.95 0.95 0.95 0.96
0.93 0.93 0.92 0.95
0.94 0.94 0.94 0.95
0.955
0.950
0.945
0.940
0.935
0.930
0.925

```
[ ]:
```