Memory_CWRU

November 16, 2023

[1]:
```
from tqdm import tqdm
import os
import pandas as pd
import numpy as np
from sklearn.svm import OneClassSVM
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px
from sklearn.preprocessing import RobustScaler
from collections import Counter
from matplotlib import pyplot as plt
plt.rcParams["figure.figsize"] = (10,10)


from sklearn.decomposition import PCA

from he_svm import preprocess_a_sample, he_svm, preprocess_a_sample_encrypted
import glob
import json
```

[3]:
```
healthy_csvs = ['data/CWRU/1_csv/Nominal/MotorLoad0.csv',
                'data/CWRU/1_csv/Nominal/MotorLoad1.csv',
                'data/CWRU/1_csv/Nominal/MotorLoad2.csv',
                'data/CWRU/1_csv/Nominal/MotorLoad3.csv']

LEN_SAMPLES = 500

train_samples = []
for f in healthy_csvs:
    df = pd.read_csv(f)
    df = df.iloc[:, :]
    dfs = df.groupby(np.arange(len(df))//LEN_SAMPLES)
    [train_samples.append(t[1]) for t in list(dfs)[:-1]]
```

[4]:
```
len(train_samples)
```

[4]: 3395

[5]:
```
len(train_samples[0])
```

[5]:
```
500
```

1 Train a SVM

[6]:
```
def preprocess_a_sample(df, windows):
    final_sample = []

    for column in df.columns:
        signal = df.loc[:, column]

        signal_fft = np.abs(np.fft.rfft(signal))**2
        len_windows = int(len(signal_fft) / windows) - 1

        for i in range(windows):
            if i == windows-1:
                final_sample.append(np.mean(signal_fft[i*len_windows:]))
            else:
                final_sample.append(np.mean(signal_fft[i*len_windows:
 ↪(i+1)*len_windows]))

    return np.array(final_sample)
```

[7]:
```
windows = 10
```

[8]:
```
preprocessed_samples_nominal = np.array([preprocess_a_sample(sample, windows)␣
 ↪for sample in train_samples])

n = int(len(preprocessed_samples_nominal) * 0.8)
preprocessed_samples_train = preprocessed_samples_nominal[:n]
preprocessed_samples_test = preprocessed_samples_nominal[n:]

svm = OneClassSVM(nu=0.05, kernel='poly', gamma='scale', degree=2)
svm.fit(preprocessed_samples_train)
svm.gamma_value = 1 / ((windows*2) * preprocessed_samples_train.var())  # to␣
 ↪put gamma value in svm
```

2 Memory occupation

[9]:
```
TRANSFER_SPEED = (1 * 1000 * 1000 * 1000) / 8  # 1 Gbit/s
```

[10]:
```
from linetimer import CodeTimer
import tenseal as ts
np.set_printoptions(precision=3, suppress=True)
```

```
poly_modulus_degree=2**14
coeff_mod_bit_sizes=[60] + [50]*6 + [60]

# Setup TenSEAL context
context = ts.context(
            ts.SCHEME_TYPE.CKKS,
            poly_modulus_degree=poly_modulus_degree,
            coeff_mod_bit_sizes=coeff_mod_bit_sizes
          )
context.generate_galois_keys()
context.global_scale = 2**50

sk = context.secret_key()

context.make_context_public()

with open('context', 'wb') as f:
    f.write(context.serialize(save_public_key=False))

file_stats = os.stat('context')

print(f'Context size in MegaBytes is {file_stats.st_size / (1024 * 1024)}')
print(f'Transfer time: {file_stats.st_size / TRANSFER_SPEED}')

os.remove('context')
```

```
Context size in MegaBytes is 349.79265117645264
Transfer time: 2.934273432
```

[11]:
```
%load_ext memory_profiler
```

[12]:
```
def fun(sample, context, windows, svm):
    x_enc_preprocessed = preprocess_a_sample_encrypted(sample, context,␣
 ↪windows, None)
    x_enc_predicted = he_svm(x_enc_preprocessed, svm, windows)
    return x_enc_predicted
```

[13]:
```
for f in ['data/CWRU/1_csv/12k_Drive/0_007_ball/Motor0.csv']:
    df = pd.read_csv(f)
    df = df.iloc[:, :]
    dfs = df.groupby(np.arange(len(df))//LEN_SAMPLES)
    anomalous_samples = [t[1] for t in list(dfs)[:-1]]

    for sample in anomalous_samples[:]:
        print(sample)
        print(f"Sample length: 2 * {len(sample)}")
```

```
        df = sample

        X = df.iloc[:, 0]
        Y = df.iloc[:, 1]
        # Z = df.loc[:, ' Z-axis']

        with CodeTimer('Encryption'):
            enc_X = ts.ckks_vector(context, X)
            enc_Y = ts.ckks_vector(context, Y)
            # enc_Z = ts.ckks_vector(context, Z)

        encrypted_sample = {'X': str(enc_X.serialize()), 'Y': str(enc_Y.
↪serialize())}

        with open('sample', 'w') as f:
            json.dump(encrypted_sample, f)


        file_stats = os.stat('sample')

        print(f'A single sample size in MegaBytes is {file_stats.st_size /␣
↪(1024 * 1024)}')
        print(f'Transfer time: {file_stats.st_size / TRANSFER_SPEED}')

        os.remove('sample')
        break
```

```
    -0.168120359281437  0.319666493506494
0             0.181278           0.326170
1             0.044345          -0.260481
2            -0.270454           0.031056
3            -0.138070           0.446980
4             0.082030           0.100485
..                 …                  …
495          -0.006660          -0.268123
496          -0.044020          -0.002114
497          -0.148628           0.074470
498          -0.051167          -0.060486
499           0.108019           0.100973

[500 rows x 2 columns]
Sample length: 2 * 500
Code block 'Encryption' took: 31.56074 ms
A single sample size in MegaBytes is 11.401249885559082
Transfer time: 0.095640616
```

[14]:
```
# Importing the library
import psutil

print('RAM Used (GB):', psutil.virtual_memory()[3]/1000000000)
```

```
RAM Used (GB): 10.531459072
```

[15]:
```
%memit res=fun(sample, context, windows, svm)
```

```
peak memory: 2677.34 MiB, increment: 1561.79 MiB
```

[16]:
```
print('RAM Used (GB):', psutil.virtual_memory()[3]/1000000000)
```

```
RAM Used (GB): 12.159873024
```

[17]:
```
res
```

[17]:
```
array([<tenseal.tensors.ckksvector.CKKSVector object at 0x7f3dcb57cfa0>],
      dtype=object)
```

[18]:
```
with open('res', 'w') as f:
    encrypted_result = {'X': str(res[0].serialize())}
    json.dump(encrypted_result, f)


file_stats = os.stat('res')

print(f'A single result in MegaBytes is {file_stats.st_size / (1024 * 1024)}')
print(f'Transfer time: {file_stats.st_size / TRANSFER_SPEED}')
```

```
A single result in MegaBytes is 0.9149236679077148
Transfer time: 0.007674936
```

[19]:
```
os.remove('res')
```

[ ]:

[ ]: