1_MSMWD_cleaning_and_extraction-ArunKumarCS

November 28, 2021

1 Microsoft Malware Detection - Cleaning and Extraction - Arun Kumar C S

The objective of this notebook is to take train.7z file and output preprocessed sparse matrices file for training.

Note: This notebook ran on 8 cpu 64 GB ram system on GCP. And the notebook has been restarted in between to clear memory for memory intensive tasks.

1.1 Downloading, unzipping and arranging files

[ ]:

```
!pip install p7zip-full
# downloaded file is train.7z
!7z x train.7z
# file extracted to train/ folder
!mkdir asmFiles
!mkdir bytesFiles
# seperating files
!mv train/*.asm asmFiles
!mv train/*.bytes bytesFiles
# Now asm files are in asmFiles folder, bytes files are in bytesFiles folder
```

[ ]:

```
!pip install nltk
```

1.2 Import

[1]:

```
import os
import time
import multiprocessing # multiprocessing

import pandas as pd
import numpy as np
import array # for pixel features

#For ngram bow
from collections import Counter
import nltk
```

```
from nltk import word_tokenize
from nltk.util import ngrams
nltk.download('punkt')

from tqdm import tqdm #visualizing progress
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /home/data_arunkumarcs/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

2 Feature Extraction

2.1 File size

[ ]:

```
# .bytes files are in bytesFile directory
# .asm files are in asmFile directory
# This is achived using "mv train/*.bytes bytesFile" and "mv train/*.asm␣
↪asmFile" commands in terminal
def get_size_dict(path, ext):
    'Gets size of all files in a directory in MB'
    fileList =os.listdir(path)
    filesize = {}
    for filename in fileList:
        filesize[filename.replace(ext,'')] = os.stat(path+"/"+filename).st_size/
↪ 1048576 # 1 mb = 1024 * 1024 bytes
    return filesize
bytesSize = get_size_dict('bytesFiles','.bytes')
asmSize = get_size_dict('asmFiles','.asm')


feature_size = pd.concat((pd.Series(asmSize),pd.Series(bytesSize)), axis = 1) \
                    .reset_index().rename(columns = {'index':'filename',0:
↪'asm_size',1:'bytes_size','Class':'Class'})
feature_size.to_csv('f_size.csv',index=False)
feature_size.head()
```

2.2 Pixel featues

[ ]:

```
files=os.listdir('asmFiles')
def get_all_px_from_asm(filename):
    # starter code: https://youtu.be/VLQTRlLGz5Y?t=847
    with open('asmFiles/'+filename,'rb') as f:
        f_size = os.path.getsize('asmFiles/'+filename)
        all_pixels = array.array('B') # uint8 array
        #https://docs.python.org/3/library/array.html -> array.fromfile(f,n)
        all_pixels.fromfile(f,10000) # we take only 10000 dim, so no need to␣
↪read all file
```

```
    #no need to reshape and save the image and open the image. First 800 pixel␣
↪intensity is already read.
    return all_pixels #returns array of 10000 pixel
px_features = {}
for i in tqdm(files):
    px_features[i.replace('.asm','')] =get_all_px_from_asm(i)
feature_pixel = pd.DataFrame(px_features).T
feature_pixel.columns = ['px_'+str(i) for i in feature_pixel.columns]
feature_pixel.to_csv('f_pixel.csv')
feature_pixel.head()
```

2.3 Asm Features

[ ]:

```
prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.
↪edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub',␣
↪'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl',␣
↪'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
keywords = ['.dll','std::',':dword']
registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']

def single_asm_counter(filename):
    asmDict = {}.fromkeys(prefixes+opcodes+keywords+registers,0)
    with open('asmFiles/'+filename, encoding = 'latin-1') as f:
        for lines in f:
            for words in lines.split():
                for p in prefixes:
                    if p in words:
                        asmDict[p]+=1
                for o in opcodes:
                    if o in words:
                        asmDict[o]+=1
                for k in keywords:
                    if k in words:
                        asmDict[k]+=1
                for r in registers:
                    if r in words:
                        asmDict[r]+=1
    filename = filename.replace('.asm','')
    return {filename:asmDict}
```

[ ]:

```
tick = time.time()
with multiprocessing.Pool(processes = 8) as pool:
    result = pool.map(single_asm_counter,files)
duration = time.time() - tick
print(duration)
```

```
# Took 136 mins
```

[ ]:

```
series_list = []
for file_idx in range(len(result)):
    filename = list(result[file_idx].keys())[0]
    series = pd.Series(result[file_idx][filename], name = filename)
    series_list.append(series)
feature_asm = pd.concat(series_list, axis=1).T
feature_asm.to_csv("f_asm.csv")
```

- We have saved all the features till now to csv files.
- Now we can restart our kernel for the next memory intensive task

3 Multiprocessing bigram bytes features in chunks

3.1 Bytes Features - unigram

[ ]:

```
# ngram using nltk: https://stackoverflow.com/questions/32441605/
 ↪generating-ngrams-unigrams-bigrams-etc-from-a-large-corpus-of-txt-files-and-t/
 ↪32442106
bytes_fn = os.listdir('bytesFiles')
def single_bytes_counter(filename,bigram=False): #switch true and false to get␣
 ↪bigram/unigram features.
    '''Used inside mp_bytes. Takes a filename, outputs {filename:ngram}
    bigram true does bigram also
    '''
    bytesText = []
    with open('bytesFiles/'+filename) as f:
        for i in f:
            bytesText.append(" ".join(i.split()[1:]))
    bytesText = " ".join(bytesText).replace('??','').strip()
    fn = filename.replace('.bytes','')
    BytesNgram = Counter(bytesText.split()) # unigram
    token = nltk.word_tokenize(bytesText)
    if bigram:
        BytesBigram = Counter([i+j for i,j in ngrams(token,2)]) #bigram features
        BytesNgram.update(BytesBigram) # unigram + bigram
    return {fn:BytesNgram}

#45 sec for 100 files
def mp_bytes(bfn):# input bytes file name list
    '''Used inside sub_counter: inputs a list of filename and does␣
 ↪multiprocessing. Outputs list of dictionary'''
    tick = time.time()
    with multiprocessing.Pool(processes = 8) as pool:
        b_result = pool.map(single_bytes_counter,bfn)
    duration = time.time() - tick
    print(duration//60,'mins')
    return b_result

def result_to_dict(r):
    '''Used inside sub_counter function'''
    r1_dic = {}
    for dic in r:
        r1_dic.update(dic)
    return r1_dic

def sub_counter(range1,range2):
    '''ngram counter for range1 to range2 and outputs a dict of {filename:␣
 ↪ngrams} in the range [range1,range2)'''
    tick = time.time()
    r = mp_bytes(bytes_fn[range1:range2])
    r = result_to_dict(r)
    print(f"{range1} to {range2} took {time.time() - tick} seconds")
    return r
```

[ ]:
```
uni_feat = sub_counter(0,len(bytes_fn))
```

[ ]:
```
uni_df = pd.DataFrame(uni_feat).T
uni_df.to_csv("f_unigram_bytes.csv")
uni_df.head()
```

3.1.1 Bytes Features - bigram

[ ]:
```
# ngram using nltk: https://stackoverflow.com/questions/32441605/
 ↪generating-ngrams-unigrams-bigrams-etc-from-a-large-corpus-of-txt-files-and-t/
 ↪32442106
bytes_fn = os.listdir('bytesFiles')
def single_bytes_counter(filename,bigram=True): #switch true and false to get␣
 ↪bigram/unigram features.
    '''Used inside mp_bytes. Takes a filename, outputs {filename:ngram}
    bigram true does bigram also
    '''
    bytesText = []
    with open('bytesFiles/'+filename) as f:
        for i in f:
            bytesText.append(" ".join(i.split()[1:]))
    bytesText = " ".join(bytesText).replace('??','').strip()
    fn = filename.replace('.bytes','')
    BytesNgram = Counter(bytesText.split()) # unigram
    token = nltk.word_tokenize(bytesText)
    if bigram:
        BytesBigram = Counter([i+j for i,j in ngrams(token,2)]) #bigram features
        BytesNgram.update(BytesBigram) # unigram + bigram
    return {fn:BytesNgram}

#45 sec for 100 files
def mp_bytes(bfn):# input bytes file name list
    '''Used inside sub_counter: inputs a list of filename and does␣
 ↪multiprocessing. Outputs list of dictionary'''
    tick = time.time()
    with multiprocessing.Pool(processes = 8) as pool:
        b_result = pool.map(single_bytes_counter,bfn)
    duration = time.time() - tick
    print(duration//60,'mins')
    return b_result

def result_to_dict(r):
    '''Used inside sub_counter function'''
    r1_dic = {}
    for dic in r:
        r1_dic.update(dic)
    return r1_dic

def sub_counter(range1,range2):
    '''ngram counter for range1 to range2 and outputs a dict of {filename:␣
 ↪ngrams} in the range [range1,range2)'''
    tick = time.time()
    r = mp_bytes(bytes_fn[range1:range2])
    r = result_to_dict(r)
    print(f"{range1} to {range2} took {time.time() - tick} seconds")
    return r
```

3.2 Multiprocessing in 4 chunks to avoid memory overflow and to checkpoint

[ ]:
```
r1= sub_counter(0,3000)
```

[ ]:
```
r1_df = pd.DataFrame(r1).T
r1_df.to_csv("feature_bytes_ngram_1.csv")
r1_df.head()
```

[ ]:
```
del r1_df # to free space in ram
del r1
r2= sub_counter(3000,6000)
```

[ ]:
```
r2_df = pd.DataFrame(r2).T
r2_df.to_csv("feature_bytes_ngram_2.csv")
```

```
r2_df.head()
```

[ ]:
```
del r2_df
del r2
r3= sub_counter(6000,9000)
```

[ ]:
```
r3_df = pd.DataFrame(r3).T
r3_df.to_csv("feature_bytes_ngram_3.csv")
r3_df.head()
```

[ ]:
```
del r3_df
del r3
r4= sub_counter(9000,len(bytes_fn))
```

[ ]:
```
r4_df = pd.DataFrame(r4).T
r4_df.to_csv("feature_bytes_ngram_4.csv")
r4_df.head()
```

- Now have all our features extracted and save into files. Now we need to combine them to a feedable format for the classifier.

3.3 We can RESTART KERNEL TO start fresh as all the features have been saved as files.

3.4 Combining chunks of bytes features

[ ]:
```
# pd.read_csv() was taking so much time. Only 12.5% of my 8 core cpu was␣
 ↪working.
# So I realized that pd.read_csv is a single thread process and the single core␣
 ↪of 8 cpu is maximum utilized.

# When I read all 4 files in parallel, cpu utilization went up nearly 50␣
 ↪percent. So I was reading the files faster.
tick = time.time()
splitted_files = ["feature_bytes_ngram_1.csv","feature_bytes_ngram_2.
 ↪csv","feature_bytes_ngram_3.csv","feature_bytes_ngram_4.csv"]
print('multiprocessing..')
with multiprocessing.Pool(processes = 4) as pool:
    read_df = pool.map(pd.read_csv,splitted_files)
duration = time.time() - tick
print(duration)
# took 5 mins to read in parallel.
```

[ ]:
```
feature_bytes = pd.concat(read_df,axis=0)
feature_bytes.head()
```

[ ]:
```
feature_bytes.to_csv("f_bigram_bytes.csv")
```

[4]:
```python
def proper_df_from_csv(file,skip_cols=0):
    """takes the file and process it for convinience
    skip_cols: skils first skip_cols number of rows"""
    features = pd.read_csv(file)
    if skip_cols:
        for i in range(skip_cols):
            print('dropping column..')
            features.drop(features.columns[0],axis=1,inplace=True)
    features = features.rename(columns={features.columns[0]:'filenames'})
    features.index = features[features.columns[0]]
    features.drop(features.columns[0],axis=1,inplace=True)
    features.sort_index(inplace=True)
    return features
```

[5]:
```python
f_unigram_bytes = proper_df_from_csv('f_unigram_bytes.csv')
f_unigram_bytes
```

[5]:
```
                            00      C1      52      02      48      25  \
filenames
01IsoiSMh5gxyDYTl4CB   39755.0  7819.0   618.0  7249.0  7011.0   301.0
01SuzwMJEIXsK7A8dQbl   19764.0   417.0   464.0   302.0   413.0   486.0
01azqd4InC7m9JpocGv5  601905.0  2997.0  3892.0  2816.0  4072.0  4002.0
01jsnpXSAlgw6aPeDxrU   93506.0  2650.0  2617.0  2568.0  2305.0  2327.0
01kcPWA9K2BOxQeS5Rju   21091.0   427.0   529.0   726.0   603.0   566.0
…                          …       …       …       …       …       …
ldNfaCpceLnGUEOrPzqF    5535.0   468.0   475.0   435.0   427.0   334.0
ljFT1KeZmEiHxhuRbrcd    4545.0   624.0   461.0   363.0   400.0   392.0
ljuryB4bfagHqV5FM9Ae    5165.0  1415.0  1295.0  1111.0  1102.0  1096.0
lkqEXK4NrYSseRTt0Gb3    5701.0   669.0   497.0   460.0   450.0   568.0
loIP1tiwELF9YNZQjSUO    5268.0  1358.0  1211.0  1072.0  1138.0  1115.0

                          C4       14      0A      31  …      B3      E1  \
filenames                                              …
01IsoiSMh5gxyDYTl4CB  8849.0  12034.0   340.0   420.0  …   432.0   400.0
01SuzwMJEIXsK7A8dQbl   303.0    437.0   237.0   321.0  …   222.0   768.0
01azqd4InC7m9JpocGv5  3280.0   3354.0  3211.0  2878.0  …  3373.0  3504.0
01jsnpXSAlgw6aPeDxrU  2318.0   2968.0  2655.0  2731.0  …  2333.0  2728.0
01kcPWA9K2BOxQeS5Rju   651.0    644.0   516.0  1047.0  …   338.0   355.0
…                        …        …       …       …  …       …       …
ldNfaCpceLnGUEOrPzqF   439.0    454.0   461.0   344.0  …   307.0   311.0
ljFT1KeZmEiHxhuRbrcd   618.0    456.0   372.0   412.0  …   372.0   407.0
ljuryB4bfagHqV5FM9Ae  1408.0   1182.0  1105.0  1146.0  …  1110.0  1080.0
lkqEXK4NrYSseRTt0Gb3   480.0    469.0   457.0   598.0  …   581.0   607.0
loIP1tiwELF9YNZQjSUO  1382.0   1111.0  1112.0  1144.0  …  1119.0  1111.0

                          C5      E7      DA      CD      A7      EE      DD  \
filenames
```

```
01IsoiSMh5gxyDYTl4CB   759.0   312.0   674.0   480.0   534.0   385.0   452.0
01SuzwMJEIXsK7A8dQbl   233.0   225.0   230.0   237.0   335.0   273.0   243.0
01azqd4InC7m9JpocGv5  2694.0  2678.0  3554.0  3068.0  2773.0  3523.0  2778.0
01jsnpXSAlgw6aPeDxrU  2360.0  2454.0  2334.0  2457.0  2584.0  2520.0  2675.0
01kcPWA9K2BOxQeS5Rju   388.0   366.0   440.0   345.0   361.0   389.0   357.0
…                        …     …     …     …     …     …     …
loIP1tiwELF9YNZQjSUO   329.0   359.0   453.0   337.0   299.0   445.0   330.0
ljFT1KeZmEiHxhuRbrcd   404.0   396.0   378.0   375.0   396.0   433.0   401.0
ljuryB4bfagHqV5FM9Ae  1133.0  1095.0  1097.0  1116.0  1029.0  1135.0  1135.0
lkqEXK4NrYSseRTt0Gb3   584.0   586.0   472.0   561.0   573.0   426.0   599.0
loIP1tiwELF9YNZQjSUO  1073.0  1119.0  1122.0  1162.0  1145.0  1128.0  1114.0

                          99
filenames
01IsoiSMh5gxyDYTl4CB   632.0
01SuzwMJEIXsK7A8dQbl   260.0
01azqd4InC7m9JpocGv5  2888.0
01jsnpXSAlgw6aPeDxrU  2529.0
01kcPWA9K2BOxQeS5Rju   357.0
…                        …
ldNfaCpceLnGUE0rPzqF   333.0
ljFT1KeZmEiHxhuRbrcd   371.0
ljuryB4bfagHqV5FM9Ae  1103.0
lkqEXK4NrYSseRTt0Gb3   549.0
loIP1tiwELF9YNZQjSUO  1105.0

[10868 rows x 256 columns]
```

[6]:
```
f_size = proper_df_from_csv('f_size.csv')
f_size
```

[6]:
```
                       asm_size  bytes_size
filenames
01IsoiSMh5gxyDYTl4CB  13.999378    6.556152
01SuzwMJEIXsK7A8dQbl   0.996723    0.438965
01azqd4InC7m9JpocGv5  56.229886    5.012695
01jsnpXSAlgw6aPeDxrU   8.507785    4.602051
01kcPWA9K2BOxQeS5Rju   0.078190    0.679688
…                           …          …
ldNfaCpceLnGUE0rPzqF   3.650518    0.467285
ljFT1KeZmEiHxhuRbrcd   4.081972    0.594727
ljuryB4bfagHqV5FM9Ae  11.279039    2.223145
lkqEXK4NrYSseRTt0Gb3   0.629995    0.835449
loIP1tiwELF9YNZQjSUO  11.269457    2.223145

[10868 rows x 2 columns]
```

[7]:
```
f_bigram_bytes = proper_df_from_csv('f_bigram_bytes.csv', skip_cols = 1)
f_bigram_bytes
```

```
dropping column..
```

[7]:
```
                            00      C1      52      02      48      25  \
filenames
01IsoiSMh5gxyDYTl4CB   39755.0  7819.0   618.0  7249.0  7011.0   301.0
01SuzwMJEIXsK7A8dQbl   19764.0   417.0   464.0   302.0   413.0   486.0
01azqd4InC7m9JpocGv5  601905.0  2997.0  3892.0  2816.0  4072.0  4002.0
01jsnpXSAlgw6aPeDxrU   93506.0  2650.0  2617.0  2568.0  2305.0  2327.0
01kcPWA9K2BOxQeS5Rju   21091.0   427.0   529.0   726.0   603.0   566.0
…                          …       …       …       …       …       …
ldNfaCpceLnGUEOrPzqF    5535.0   468.0   475.0   435.0   427.0   334.0
ljFT1KeZmEiHxhuRbrcd    4545.0   624.0   461.0   363.0   400.0   392.0
ljuryB4bfagHqV5FM9Ae    5165.0  1415.0  1295.0  1111.0  1102.0  1096.0
lkqEXK4NrYSseRTtOGb3    5701.0   669.0   497.0   460.0   450.0   568.0
loIP1tiwELF9YNZQjSUO    5268.0  1358.0  1211.0  1072.0  1138.0  1115.0

                          C4       14      0A      31  …  2842  5495  B32E  \
filenames                                               …
01IsoiSMh5gxyDYTl4CB  8849.0  12034.0   340.0   420.0  …   NaN   1.0   NaN
01SuzwMJEIXsK7A8dQbl   303.0    437.0   237.0   321.0  …   NaN   7.0   3.0
01azqd4InC7m9JpocGv5  3280.0   3354.0  3211.0  2878.0  …   6.0   4.0  13.0
01jsnpXSAlgw6aPeDxrU  2318.0   2968.0  2655.0  2731.0  …   3.0   6.0   5.0
01kcPWA9K2BOxQeS5Rju   651.0    644.0   516.0  1047.0  …   NaN   3.0   NaN
…                        …        …       …       …  …     …     …     …
ldNfaCpceLnGUEOrPzqF   439.0    454.0   461.0   344.0  …   1.0   NaN   NaN
ljFT1KeZmEiHxhuRbrcd   618.0    456.0   372.0   412.0  …   2.0   3.0   NaN
ljuryB4bfagHqV5FM9Ae  1408.0   1182.0  1105.0  1146.0  …   6.0   6.0   5.0
lkqEXK4NrYSseRTtOGb3   480.0    469.0   457.0   598.0  …   NaN   1.0   2.0
loIP1tiwELF9YNZQjSUO  1382.0   1111.0  1112.0  1144.0  …   5.0   2.0   5.0

                      6596  E55A  41CA  A014  AE08  5D91  55F2
filenames
01IsoiSMh5gxyDYTl4CB   1.0   NaN   NaN   NaN   NaN   1.0   NaN
01SuzwMJEIXsK7A8dQbl   NaN   2.0   3.0   2.0   2.0   NaN   NaN
01azqd4InC7m9JpocGv5   8.0   6.0   6.0  13.0   3.0   5.0   4.0
01jsnpXSAlgw6aPeDxrU   4.0   2.0   9.0   6.0  12.0   3.0   4.0
01kcPWA9K2BOxQeS5Rju   2.0   1.0   3.0   2.0   NaN   NaN   4.0
…                    …     …     …     …     …     …     …
ldNfaCpceLnGUEOrPzqF   1.0   1.0   1.0   NaN   1.0   NaN   1.0
ljFT1KeZmEiHxhuRbrcd   NaN   1.0   2.0   3.0   NaN   1.0   1.0
ljuryB4bfagHqV5FM9Ae   5.0   3.0   3.0   3.0   5.0   4.0   3.0
lkqEXK4NrYSseRTtOGb3   3.0   NaN   1.0   3.0   3.0   1.0   3.0
loIP1tiwELF9YNZQjSUO   3.0   8.0   5.0   5.0   7.0   3.0   5.0
```

```
[10868 rows x 65792 columns]
```

[8]:
```
f_pixel = proper_df_from_csv('f_pixel.csv')
f_pixel
```

[8]:
```
                      px_0  px_1  px_2  px_3  px_4  px_5  px_6  px_7  px_8  \
filenames
01IsoiSMh5gxyDYTl4CB    46   116   101   120   116    58    48    48    52
01SuzwMJEIXsK7A8dQbl    72    69    65    68    69    82    58    48    48
01azqd4InC7m9JpocGv5    72    69    65    68    69    82    58    48    48
01jsnpXSAlgw6aPeDxrU    72    69    65    68    69    82    58    48    48
01kcPWA9K2BOxQeS5Rju    72    69    65    68    69    82    58    49    48
…                    …   …    …    …   …    …    …    …   …
ldNfaCpceLnGUEOrPzqF    72    69    65    68    69    82    58    49    48
ljFT1KeZmEiHxhuRbrcd    72    69    65    68    69    82    58    49    48
ljuryB4bfagHqV5FM9Ae    72    69    65    68    69    82    58    49    48
lkqEXK4NrYSseRTt0Gb3    72    69    65    68    69    82    58    49    48
loIP1tiwELF9YNZQjSUO    72    69    65    68    69    82    58    49    48

                      px_9  …  px_9990  px_9991  px_9992  px_9993  px_9994  \
filenames                   …
01IsoiSMh5gxyDYTl4CB    48  …       58       48       48       52       48
01SuzwMJEIXsK7A8dQbl    52  …       67      104       44       32       50
01azqd4InC7m9JpocGv5    52  …      100      119      111      114      100
01jsnpXSAlgw6aPeDxrU    52  …       48       70       70       48       48
01kcPWA9K2BOxQeS5Rju    48  …       48        9        9        9        9
…                    …  …       …      …       …       …       …
ldNfaCpceLnGUEOrPzqF    48  …       32       50       70        9        9
ljFT1KeZmEiHxhuRbrcd    48  …      116      101      120      116       58
ljuryB4bfagHqV5FM9Ae    48  …       32       50       55      104       44
lkqEXK4NrYSseRTt0Gb3    48  …        9        9        9        9       32
loIP1tiwELF9YNZQjSUO    48  …       49       49       48       50       32

                      px_9995  px_9996  px_9997  px_9998  px_9999
filenames
01IsoiSMh5gxyDYTl4CB       49       49       54       56       32
01SuzwMJEIXsK7A8dQbl       55       56       52       65       52
01azqd4InC7m9JpocGv5       95       53       54       50       52
01jsnpXSAlgw6aPeDxrU       66       55       48       48      104
01kcPWA9K2BOxQeS5Rju        9       32       32       32       32
…                        …       …       …       …       …
ldNfaCpceLnGUEOrPzqF        9        9        9        9        9
ljFT1KeZmEiHxhuRbrcd       49       48       48       48       49
ljuryB4bfagHqV5FM9Ae       32       97      108       13       10
lkqEXK4NrYSseRTt0Gb3       32       32       32       32       32
loIP1tiwELF9YNZQjSUO       53       51        9        9        9
```

```
[10868 rows x 10000 columns]
```

[9]:

```python
f_asm = proper_df_from_csv('f_asm.csv')
f_asm
```

[9]:

```
                       HEADER:  .text:  .Pav:  .idata:   .data:  .bss:  \
filenames
01IsoiSMh5gxyDYTl4CB        0  110032      0      616    24618      0
01SuzwMJEIXsK7A8dQbl       26   10456      0      206     4686     96
01azqd4InC7m9JpocGv5       24   23226      0     1158  1366755      0
01jsnpXSAlgw6aPeDxrU       22   68915      0      304      662      0
01kcPWA9K2BOxQeS5Rju       24     782      0      127       58      0
…                         …       …      …       …        …      …
ldNfaCpceLnGUEOrPzqF       23    4490      0        3    31906      0
ljFT1KeZmEiHxhuRbrcd       25     320      0      106    75792      0
ljuryB4bfagHqV5FM9Ae       24     436      0      130   263653      0
lkqEXK4NrYSseRTtOGb3       24    2840      0        0    10919      0
loIP1tiwELF9YNZQjSUO       24     640      0      109   264208      0

                       .rdata:  .edata:  .rsrc:  .tls:  …  :dword   edx  \
filenames                                                 …
01IsoiSMh5gxyDYTl4CB     26760        0       0      0  …     227   724
01SuzwMJEIXsK7A8dQbl         0        0       3      0  …      76  1121
01azqd4InC7m9JpocGv5      2263        0       0      0  …     456  1490
01jsnpXSAlgw6aPeDxrU      1236        0       0      0  …     117   525
01kcPWA9K2BOxQeS5Rju       381        0       3      0  …      29    23
…                          …       …       …      …   …      …     …
ldNfaCpceLnGUEOrPzqF     55313        3       3      0  …       0   363
ljFT1KeZmEiHxhuRbrcd     24591        0       3      0  …      41     0
ljuryB4bfagHqV5FM9Ae       386        0       3      0  …      53   173
lkqEXK4NrYSseRTtOGb3         0        0       0      0  …       0   189
loIP1tiwELF9YNZQjSUO       323        0       3      0  …      42   153

                       esi   eax  ebx   ecx   edi   ebp  esp  eip
filenames
01IsoiSMh5gxyDYTl4CB   502  1446  260  1090   391   905  420    0
01SuzwMJEIXsK7A8dQbl    28  1220   18  1228    24  1546  107    0
01azqd4InC7m9JpocGv5  1898  4371  808  2290  1281   587  701    0
01jsnpXSAlgw6aPeDxrU     6   903    5   547     5   451   56    0
01kcPWA9K2BOxQeS5Rju    35   137   18    66    15    43   83    0
…                     …    …    …    …    …    …    …   …
ldNfaCpceLnGUEOrPzqF   340   473  294   462   273   228  287    0
ljFT1KeZmEiHxhuRbrcd     1     6    2     1     2     0    0    0
ljuryB4bfagHqV5FM9Ae   247   345  218   207   162   132  154    0
lkqEXK4NrYSseRTtOGb3   153   182   96    69    72    33  134    0
loIP1tiwELF9YNZQjSUO   184   318  193   177    99   103  134    0
```

```
[10868 rows x 51 columns]
```

[29]:

```
target = proper_df_from_csv('trainLabels.csv')
target
```

[29]:

```
                      Class
filenames
01IsoiSMh5gxyDYTl4CB      2
01SuzwMJEIXsK7A8dQbl      8
01azqd4InC7m9JpocGv5      9
01jsnpXSAlgw6aPeDxrU      9
01kcPWA9K2BOxQeS5Rju      1
…                     …
ldNfaCpceLnGUEOrPzqF      4
ljFT1KeZmEiHxhuRbrcd      4
ljuryB4bfagHqV5FM9Ae      4
lkqEXK4NrYSseRTt0Gb3      4
loIP1tiwELF9YNZQjSUO      4

[10868 rows x 1 columns]
```

4 Check and replace null values

[10]:

```
np.any(f_size.isnull().values)
```

[10]:

```
False
```

[11]:

```
np.any(f_asm.isnull().values)
```

[11]:

```
False
```

[12]:

```
np.any(f_unigram_bytes.isnull().values)
```

[12]:

```
True
```

[13]:

```
f_unigram_bytes.fillna(0, inplace=True)
```

[14]:

```
np.any(f_unigram_bytes.isnull().values)
```

[14]:

```
False
```

[15]:

```
np.any(f_pixel.isnull().values)
```

[15]:

```
False
```

[16]:

```
np.any(f_bigram_bytes.isnull().values)
```

[16]: True

```
[17]: f_bigram_bytes.fillna(0, inplace=True)
```

```
[18]: np.any(f_bigram_bytes.isnull().values)
```

[18]: False

5 Allign the feature indices to stack together.

```
[30]: for i in [f_asm,f_unigram_bytes, f_pixel,f_bigram_bytes, target]:
          print(all(f_size.index == i.index)) # Checking if all the indexes are alligned for stacking
```

```
True
True
True
True
True
```

5.1 Save as numpy arrays

```
[23]: np.save('f_size.npy',f_size.values)
      np.save('f_asm.npy',f_asm.values)
      np.save('f_unigram_bytes.npy',f_unigram_bytes.values)
```

```
[24]: np.save('f_pixel.npy',f_pixel.values)
```

```
[25]: np.save('f_bigram_bytes.npy',f_bigram_bytes.values)
```

```
[31]: np.save('target.npy',target)
```

All our data is in preprocessed and stored in f_feature_name.npy files. Only this need to be imported for training.