Microsoft Malware detection

Business/Real-world Problem

What is Malware?

The term malware is a contraction of malicious software. Put simply, malware is any piece of software that was written with the intent of doing harm to data, devices or to people.
Source: https://www.avg.com/en/signal/what-is-malware

Problem Statement

In the past few years, the malware industry has grown very rapidly that, the syndicates invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust softwares to detect and terminate these attacks. The major part of protecting a computer system from a malware attack is to **identify whether a given piece of file/software is a malware.**

1.3 Source/Useful Links

Microsoft has been very active in building anti-malware products over the years and it runs it's anti-malware utilities over 150 million computers around the world. This generates tens of millions of daily data points to be analyzed as potential malware. In order to be effective in analyzing and classifying such large amounts of data, we need to be able to group them into groups and identify their respective families.

This dataset provided by Microsoft contains about 9 classes of malware. ,

Source: https://www.kaggle.com/c/malware-classification

1.4. Real-world/Business objectives and constraints.

1. **Minimize multi-class error.**
2. **Multi-class probability estimates.**
3. **Malware detection should not take hours and block the user's computer. It should fininsh in a few seconds or a minute.**

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- **Source : https://www.kaggle.com/c/malware-classification/data**
- **For every malware, we have two files**
  1. **.asm file (read more: https://www.reviversoft.com/file-extensions/asm)**
  2. **.bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)**
- **Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:**
- **Lots of Data for a single-box/computer.**
- **There are total 10,868 .bytes files and 10,868 asm files total 21,736 files**
- **There are 9 types of malwares (9 classes) in our give data**

- There are 9 types of malwares (9 classes) in our give data
- **Types of Malware:**
  1. **Ramnit**
  2. **Lollipop**
  3. **Kelihos_ver3**
  4. **Vundo**
  5. **Simda**
  6. **Tracur**
  7. **Kelihos_ver1**
  8. **Obfuscator.ACY**
  9. **Gatak**

2.1.2. Example Data Point

** .asm file **

```
.text:00401000                                  assume es:nothing, ss:nothing, ds:_data, fs:n
othing, gs:nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24 08                          lea     eax, [esp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                                mov     esi, ecx
.text:00401008 E8 1C 1B 00 00                           call    ??0exception@std@@QAE@ABQBD@Z
; std::exception::exception(char const * const &)
.text:0040100D C7 06 08 BB 42 00                        mov     dword ptr [esi], offset off_4
2BB08
.text:00401013 8B C6                                mov     eax, esi
.text:00401015 5E                               pop     esi
.text:00401016 C2 04 00                             retn    4
.text:00401016                          ; ---------------------------------------------------
------------------------
.text:00401019 CC CC CC CC CC CC CC                     align 10h
.text:00401020 C7 01 08 BB 42 00                        mov     dword ptr [ecx], offset off_4
2BB08
.text:00401026 E9 26 1C 00 00                           jmp     sub_402C51
.text:00401026                          ; ---------------------------------------------------
------------------------
.text:0040102B CC CC CC CC CC                           align 10h
.text:00401030 56                               push    esi
.text:00401031 8B F1                                mov     esi, ecx
.text:00401033 C7 06 08 BB 42 00                        mov     dword ptr [esi], offset off_4
2BB08
.text:00401039 E8 13 1C 00 00                           call    sub_402C51
.text:0040103E F6 44 24 08 01                           test    byte ptr [esp+8], 1
.text:00401043 74 09                                jz      short loc_40104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E 00 00                           call    ??3@YAXPAX@Z    ; operator de
lete(void *)
.text:0040104B 83 C4 04                             add     esp, 4
.text:0040104E
.text:0040104E                          loc_40104E:                 ; CODE XREF: .text:004010
43j
.text:0040104E 8B C6                                mov     eax, esi
.text:00401050 5E                               pop     esi
.text:00401051 C2 04 00                             retn    4
.text:00401051                          ; ---------------------------------------------------
------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 00 20 00 02 00 04
```

```
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
```

Mapping the real-world problem to an ML problem

Type of Machine Learning Problem

There are nine different classes of malware that we need to classify a given a data point => Multi class classification problem

Performance Metric

Source: https://www.kaggle.com/c/malware-classification#evaluation

Metric(s):

- **Multi class log-loss**
- **Confusion matrix**

Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- **Class probabilities are needed.**
- **Penalize the errors in class probabilites => Metric is Log-loss.**
- **Some Latency constraints.**

Reference blogs, videos and reference papers

- **Reference for dask parallel implementation link1 link2**
- **http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/**
- **https://arxiv.org/pdf/1511.04317.pdf**
- **First place solution in Kaggle competition**
- **https://github.com/dchad/malware-detection**
- **http://vizsec.org/files/2011/Nataraj.pdf**
- **https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EeInEjvvuQg2nu_pIB6ua?dl=0**
- **"Cross validation is more trustworthy than domain knowledge."**

File Strucure:

- **MMD1:**
  1. **Read Data**
  2. **Exploratoray Data Analysis**
  3. **Feature Extraction**
- **MMD2:**

    1. Feature Selection
    2. Validating selected features
    3. Final Data set preperation
- MMD3:

    1. Train Test Split
    2. Modelling And HyperParam Tuning
    3. Compute performance on test data

Machine:

- Run time -
- Intel 12 Cores (6 CPU)
- 16GB RAM
- 256GB SSD
- 1TB HDD

In this Notebook I tried various ways suggestted by other kaggle winners and Srikanth sir in case study videos to select best features based on their importances. I have referenced the code blocks which I have taken from other sources. I have tried various time complexity reduction techniques (using various parallelizing techniques) and equipped the best working ones according to my system specs.

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
# from multiprocessing import Process# this is used for multithreading
# import multiprocessing
import codecs# this is used for file operations
import random as r
from scipy import sparse as sp
```

In [2]:

```
import os
os.chdir("D:/LargeDatasets/MicrosoftMalware/")
```

In [3]:

```
from dask import delayed
import dask.array as da
import dask.bag as db
import dask.dataframe as ddf
```

In [4]:

```
from tqdm import tqdm
tqdm.pandas()
import pickle as pk
import hickle as hk
import joblib as jb
import random
import string
import array
import math
import sys
```

```
import gc
```

In [5]:

```
from dask.diagnostics import ProgressBar
ProgressBar().register()
```

In [6]:

```
# from dask.distributed import Client,progress
# client = Client(n_workers=5)
```

In [7]:

```
# client
```

In [8]:

```
# separating byte files and asm files

# source = 'train'
# destination = 'byteFiles'

# # we will check if the folder 'byteFiles' exists if it not there we will create a folder with the sam
e name
# if not os.path.isdir(destination):
#     os.makedirs(destination)

# if os.path.isdir(source):
#     os.rename(source,'asmFiles')
#     source='asmFiles'
#     data_files = os.listdir(source)
```

In [9]:

```
# def separate_files(file):
#     source='asmFiles/'
#     destination = 'byteFiles'
#     if (file.endswith("bytes")):
#         shutil.move(source+file,destination)
# b = db.from_sequence(data_files,npartitions=50)
# b.map(separate_files).compute()
```

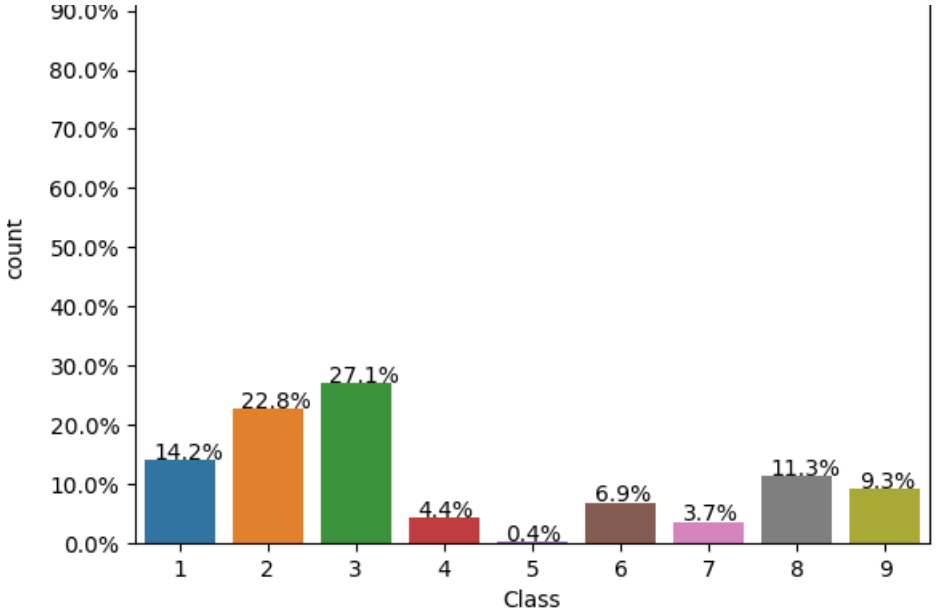
3.1. Distribution of malware classes in whole data set

In [6]:

```
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI
Y=pd.read_csv("trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

#put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
ax.yaxis.set_ticks(np.linspace(0, total, 11))

#adjust the ticklabel to the desired format, without changing the position of the ticks.
ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
plt.show()
```



Feature extraction from byte files

File size of byte files as a feature

In [7]:

```
#file sizes of byte files
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI

hexdumps = os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()

def compute_size(file,files=filenames,labels=class_y):
    statinfo=os.stat('byteFiles/'+file)
    file=file.split('.')[0]
    if file in files:
        i=files.index(file)
        return (file,statinfo.st_size/(1024.0*1024.0),class_y[i])

a = db.from_sequence(hexdumps,npartitions=50)
fnames,sizebytes,class_bytes = zip(*a.map(compute_size).compute())

data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
data_size_byte.head()
```

```
[########################################] | 100% Completed |  3.6s
```

Out[7]:

	ID	size	Class
0	01azqd4InC7m9JpocGv5	4.234863	9
1	01IsoiSMh5gxyDYTl4CB	5.538818	2
2	01jsnpXSAlgw6aPeDxrU	3.887939	9
3	01kcPWA9K2BOxQeS5Rju	0.574219	1
4	01SuzwMJEIXsK7A8dQbl	0.370850	8

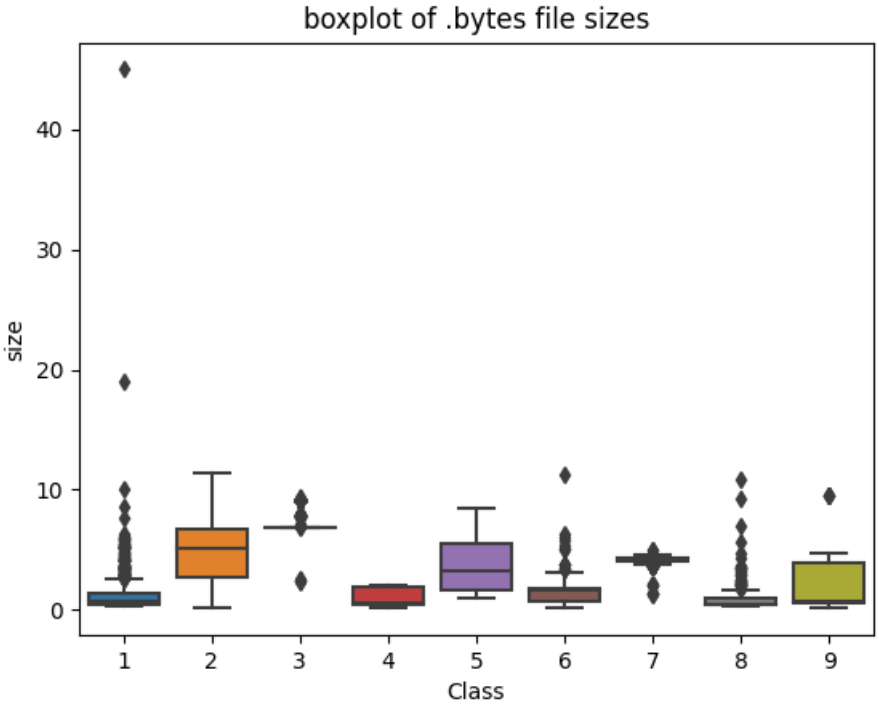
3.2.2 box plots of file size (.byte files) feature

In [12]:

```
# data_size_byte.describe()
```

In [13]:

```
#boxplot of byte files
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



3.2.3 feature extraction from byte files

In [8]:

```
byte_features = ["ID"]
for i in range(256):
    byte_features.append(hex(i)[2:])
byte_features.append("??")
byte_features = np.array(byte_features)
```

In [9]:

```
hexdumps_paths = list(map(lambda x:'byteFiles/'+x,hexdumps))
```

In [10]:

```
# @delayed
# def remove_id(file,ext):
#     text_file = open('byteFiles/'+file+".txt", 'w+')
#     with open('byteFiles/'+file+'.'+ext,"r") as fp:
#         lines=""
#         for line in fp:
#             a=line.rstrip().split(" ")[1:]
#             b=' '.join(a)
#             b=b+"\n"
#             text_file.write(b)
#         fp.close()
#         os.remove('byteFiles/'+file+'.'+ext)
#     return text_file
```

In [11]:

```
# Ref: https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.ipynb
def generate_one_gram(file):
    '''Generates 1-gram byte features'''
    # create a empty numpy vector
    text_file, ID = file
    ID=os.path.basename(ID).split('.')[0]
    one_gram = np.zeros((1,257),dtype=int)

    for lines in text_file:
        line=lines[:-1].rstrip().split(" ")
        ## Generate Uni-gram
        for hex_code in line:
            if hex_code=='??':
                one_gram[0][256]+=1
            else:
                one_gram[0][int(hex_code,16)]+=1
    return np.append(np.array(ID),one_gram)

#Ref: https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.ipynb
def generate_two_gram(file):
    '''Generates 2-gram byte features'''
    text_file, ID = file
    ID=os.path.basename(ID).split('.')[0]
    two_gram = np.zeros((1,16**4),dtype=int)

    for lines in text_file:
        line=lines[:-1].rstrip().split(" ")
        two_gram_line = line[:-1]
        for i in range(len(two_gram_line)):
            two_gram_line[i]+=line[i+1]
        for hex_code in two_gram_line:
            if '??' not in hex_code:
                two_gram[0,int(hex_code,16)]+=1
#     f_name = ''.join(random.choices(string.ascii_uppercase +
#                              string.digits, k = 7))
    res = np.append(ID,two_gram)
    hk.dump(res,'twoGram/'+ID+'.hkl',mode='w',compression='gzip')
    res = None
    two_gram = None
    return

# This is my idea, not referred from anywhere
def generate_four_gram_hash(file):
    '''Generates 4-gram i.e,8bit hex reduced to 4bit byte features using hashing. 65535^2 features are
reduced to 65535 features through hashing.'''
    text_file, ID = file
    ID=os.path.basename(ID).split('.')[0]
    four_gram = np.zeros((1,16**4),dtype=int)

    for lines in text_file:
        line=lines[:-1].rstrip().split(" ")
        two_gram_line = line[:-1]
        four_gram_line = two_gram_line[:-2]
        for i in range(len(four_gram_line)):
            four_gram_line[i]+=two_gram_line[i+2]
        for hex_code in four_gram_line:
            if '??' not in hex_code:
                four_gram[0,int(hex_code,16)%65536]+=1
    res = np.append(ID,four_gram)
    hk.dump(res,'hashFourGram/'+ID+'.hkl',mode='w',compression='gzip')
    res=None
    four_gram=None
    return

# ref: https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.ipynb
def generate_four_gram(file):
    '''Generates 4-gram byte features using dask arrays'''
    text_file, ID = file
    ID=os.path.basename(ID).split('.')[0]
    four_gram = sp.csc_matrix((1,16**8),dtype=int)

    for lines in text_file:
        line=lines[:-1].rstrip().split(" ")
```

```
        two_gram_line = line[:-1]
        four_gram_line = two_gram_line[:-2]
        for i in range(len(four_gram_line)):
            four_gram_line[i]+=two_gram_line[i+2]
        for hex_code in four_gram_line:
            if '??' not in hex_code:
                four_gram[0,int(hex_code,16)%65536]+=1
    return sp.hstack((sp.csc_matrix(ID),four_gram))

# def chunk_no_generator():
#     '''Generate unique id for each hickle file of each partition dumped'''
#     for  i in range(10868):
#         yield str(i)
# @delayed
def compute_byte_features(files,n_gram,hsh=True):
    '''Generates count byte features'''
    chunk=[]
    for path,file in files:
        chunk.append((file.compute(),path))
    if n_gram==1:
        res = [generate_one_gram(file) for file in chunk]
    elif n_gram==2:
#         print("Computing {}-gram features".format(n_gram))
        for file in chunk:
            generate_two_gram(file)
        res = True
    elif n_gram==4 and hsh==True:
        for file in chunk:
            generate_four_gram_hash(file)
        res = True
    elif n_gram==4 and hsh==False:
        res = [generate_four_gram(file) for file in chunk]
    else:
        raise valueError("Entered Unsupported function arguments!!")

    ## Ensuring chunk is no longer in memmory to avoid memmory overflow.
    del chunk
    gc.collect()
    return res

# @delayed
# def merge_size_features(var1,var2):
#     return np.vstack((var1,var2))
```

In [40]:

```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            df[feature_name] = df[feature_name].apply(float)
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
# result = normalize(result)
```

One Grams

In [13]:

```
# %%time
one_grams = db.from_sequence(list(zip(hexdumps_paths,db.read_text(hexdumps_paths,collection=False))),np
artitions=209)
```

In [14]:

```
# %%time
one_grams = one_grams.map_partitions(compute_byte_features,n_gram=1).compute()
one_gram_features = np.vstack(one_grams)
print("Done!")
```

```
[########################################] | 100% Completed | 55min 37.9s
Done!
```

In [10]:

```
# one_gram_features = pd.DataFrame(one_gram_features,columns=byte_features)
# one_gram_features = pd.merge(one_gram_features, data_size_byte,on='ID', how='left')
# one_gram_features.to_csv("one_gram_byte_features.csv",index=False)
# print("Done!!")
```

In [6]:

```
one_gram_features = pd.read_csv("one_gram_byte_features.csv")
# one_gram_features.drop(["Unnamed: 0"],axis=1,inplace=True)
# one_gram_features = None
one_gram_features.head()
```

Out[6]:

	ID	0	1	2	3	4	5	6	7	8	...	fa	fb	fc	fd	fe	ff
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3211	3097	2758	3099	2759	5753
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	281	302	7639	518	17001	54902
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2885	2863	2471	2786	2680	49144
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	462	516	1133	471	761	7998
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	209	239	653	221	242	2199

5 rows × 261 columns

One Gram Byte Entropy

REF: Link Here

In [7]:

```
def count_entropy(byte_counts):
    res = 0.0
    tot = sum(byte_counts)
    for count in byte_counts:
        if count==0:
            continue
        p = 1*count/tot
        res -= p*math.log(p,10)
    return res
```

In [8]:

```
one_gram_features["byte_entropy"] = one_gram_features.iloc[:,1:-2].progress_apply(count_entropy,axis=1)
```

```
100%|██████████| 10868/10868 [00:01<00:00, 6454.64it/s]
```

In [9]:

```
one_gram_features.head()
```

Out[9]:

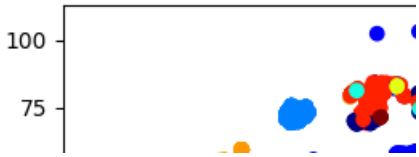
	ID	0	1	2	3	4	5	6	7	8	...	fa	fb	fc	fd	fe	ff
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3211	3097	2758	3099	2759	5753
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	281	302	7639	518	17001	54902
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2885	2863	2471	2786	2680	49144

	ID	0	1	2	3	4	5	6	7	8	...	fa	fb	fc	fd	fe	ff
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	462	516	1133	471	761	7998
4	01SuzwMJEXsK7A8dQbI	19764	710	302	433	559	410	262	249	422	...	209	239	653	221	242	2199

5 rows × 261 columns

In [7]:

```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(one_gram_features.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=one_gram_features["Class"], cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



Two Grams

In [ ]:

```
%%time
two_grams = db.from_sequence(list(zip(hexdumps_paths,db.read_text(hexdumps_paths,collection=False))),np
artitions=572)
```

In [ ]:

```
if not os.path.isdir("twoGram"):
    os.mkdir("twoGram")
```

In [ ]:

```
%%time
# gen = chunk_no_generator()
two_grams = two_grams.map_partitions(compute_byte_features,n_gram=2).compute()
# two_gram_features = np.vstack(two_grams)
print("Done!")
```

In [ ]:

```
two_gram_hkl_files = list(map(lambda x:'twoGram/'+x,os.listdir('twoGram/')))
```

In [ ]:

```
@delayed
def load_hkl(file):
    return hk.load(file)
```

In [ ]:

```
two_gram_objs = jb.Parallel(n_jobs=-2,verbose=3)(jb.delayed(load_hkl)(file) for file in (two_gram_hkl_f
iles))
```

In [ ]:

```
def merge_hkls(files):
    ar=[]
    for file in files:
        ar.append(file.compute())
    ar = np.vstack(ar)
    f_name = (''.join(random.choices(string.ascii_uppercase +
                        string.digits, k = 7)))+'.csv'
    pd.DataFrame(ar).to_csv(f_name)
    ar = None
    return f_name
```

In [ ]:

```
%%time
os.chdir("./twoGram")
two_gram_objs = db.from_sequence(two_gram_objs,npartitions=100)
two_gram_csv_files = two_gram_objs.map_partitions(merge_hkls).compute()
os.chdir("../")
```

In [ ]:

```
## Remove hkl files
# map(lambda file:os.remove(file),two_gram_hkl_files)
```

In [ ]:

```
col = ["ID"]
for i in range(65536):
    col.append(str(i))
```

In [ ]:

```
%%time
two_gram_csv_files = jb.Parallel(n_jobs=-2,verbose=2,prefer="processes")(jb.delayed(lambda file:pd.read
_csv('./twoGram/'+file))(file) for file in two_gram_csv_files)
two_gram_features = pd.concat(two_gram_csv_files,ignore_index=True)
```

In [ ]:

```
two_gram_features.set_index(two_gram_features.columns[0],inplace=True)
two_gram_features.columns = col
two_gram_features.head()
```

In [ ]:

```
%%time
two_gram_features = pd.merge(two_gram_features, data_size_byte,on='ID', how='left')
two_gram_features.to_csv("two_gram_byte_features.csv")
```

In [ ]:

```
two_gram_features.head()
```

In [ ]:

```
# %%timea
# # client.compute()
# results = byte_features.result()
# while len(results)>1:
#     temp = []
#     n=len(results)
#     for i in range(0,n,2):
#         if i!=n-1:
#             res = merge_size_features(results[i],results[i+1])
#             temp.append(res)
#         else:
#             temp.append(results[i])
#     results = temp
```

```
# byte_features = results[0]
```

Four Grams Hash Encoded

In [ ]:

```
%%time
four_grams = db.from_sequence(list(zip(hexdumps_paths,db.read_text(hexdumps_paths,collection=False))),n
partitions=572)
```

In [ ]:

```
if not os.path.isdir("hashFourGram"):
    os.mkdir("hashFourGram")
```

In [ ]:

```
%%time
# gen = chunk_no_generator()
four_grams = two_grams.map_partitions(compute_byte_features,n_gram=4,hsh=True).compute()
# four_gram_features = np.vstack(four_grams)
print("Done!")
```

In [ ]:

```
four_gram_hsh_hkl_files = os.listdir('hashFourGram/')
```

In [ ]:

```
four_gram_hsh_objs = jb.Parallel(n_jobs=-2,verbose=2)(jb.delayed(load_hkl)(file) for file in (four_gram
_hsh_hkl_files))
```

In [ ]:

```
%%time
os.chdir("./hashFourGram")
four_gram_hsh_objs = db.from_sequence(four_gram_hsh_objs,npartitions=100)
four_gram_hsh_csv_files = four_gram_hsh_objs.map_partitions(merge_hkls).compute()
os.chdir("../")
```

In [ ]:

```
%%time
four_gram_hsh_csv_files = jb.Parallel(n_jobs=-2,verbose=2,prefer="processes")(jb.delayed(lambda file:pd
.read_csv('./hashFourGram/'+file))(file) for file in four_gram_hsh_csv_files)
four_gram_hsh_features = pd.concat(four_gram_hsh_csv_files)
```

In [ ]:

```
## Remove hkl files
# map(lambda file:os.remove(file),two_gram_hkl_files)
```

In [ ]:

```
four_gram_hsh_features.set_index(four_gram_hsh_features.columns[0],inplace=True)
four_gram_hsh_features.columns = col
four_gram_hsh_features.head()
```

In [ ]:

```
%%time
four_gram_hsh_features = pd.merge(four_gram_hsh_features, data_size_byte,on='ID', how='left')
four_gram_hsh_features.to_csv("fou_gram_hash_encoded_byte_features.csv")
```

Feature extraction from asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In parallel we can use all th
e cores that are present in our computer.

Here we extracted 52 features from all the asm files which are important.

We read the top solutions and handpicked the features from those papers/videos/blogs.
Refer:https://www.kaggle.com/c/malware-classification/discussion

In [7]:

```
#file sizes of asm files
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI

asm_files = os.listdir('asmFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()

def compute_size(file,files=filenames,labels=class_y):
    statinfo=os.stat('asmFiles/'+file)
    file=file.split('.')[0]
    if file in files:
        i=files.index(file)
        return (file,statinfo.st_size/(1024.0*1024.0),class_y[i])

a = db.from_sequence(asm_files,npartitions=50)
fnames,sizeasm,class_asm = zip(*a.map(compute_size).compute())

data_size_asm=pd.DataFrame({'ID':fnames,'size':sizeasm,'Class':class_asm})
data_size_asm.head()
```

```
[########################################] | 100% Completed |  3.4s
```

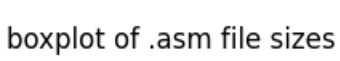
Out[7]:

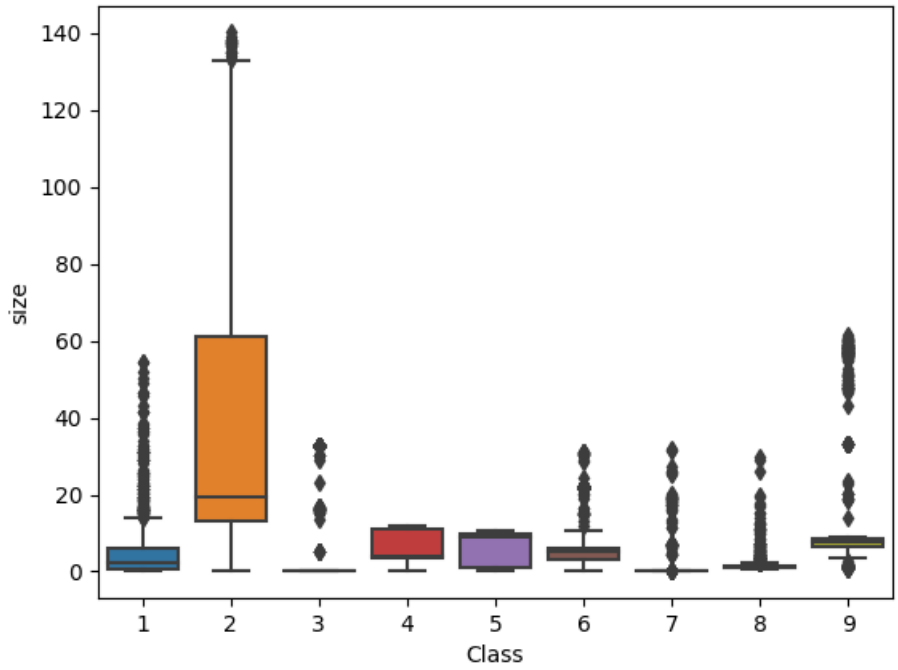
	ID	size	Class
0	01azqd4InC7m9JpocGv5	56.229886	9
1	01IsoiSMh5gxyDYTl4CB	13.999378	2
2	01jsnpXSAlgw6aPeDxrU	8.507785	9
3	01kcPWA9K2BOxQeS5Rju	0.078190	1
4	01SuzwMJEIXsK7A8dQbl	0.996723	8

In [8]:

```
#boxplot of byte files
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI

ax = sns.boxplot(x="Class", y="size", data=data_size_asm)
plt.title("boxplot of .asm file sizes")
plt.show()
```





In [15]:

```
@delayed
def load_asm(file):
    return codecs.open(file,encoding='cp1252',errors ='replace')

# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI
def asm_features_util(file,prefixes,opcodes,registers,keywords):
    '''Compute prefix counts,opcodes counts,keyword counts, and registers counts for input files.'''
    asm_file, ID = file
    ID=os.path.basename(ID).split('.')[0]

    prefixes_count=np.zeros(len(prefixes),dtype=int)
    opcodes_count=np.zeros(len(opcodes),dtype=int)
    keyword_count=np.zeros(len(keywords),dtype=int)
    registers_count=np.zeros(len(registers),dtype=int)
    features=[]
#     f2=file.split('.')[0]
    for line in asm_file:
#         print(lines)
#         print("\n\n")
        line=line.rstrip().split()
        l=line[0]
        for i in range(len(prefixes)):
            if prefixes[i] in line[0]:
                prefixes_count[i]+=1
        line=line[1:]
        for i in range(len(opcodes)):
            if opcodes[i] in line:
                features.append(opcodes[i])
                opcodes_count[i]+=1
        for i in range(len(registers)):
            for li in line:
                if registers[i] in li and ('text' in l or 'CODE' in l):
                    registers_count[i]+=1
        for i in range(len(keywords)):
            for li in line:
                if keywords[i] in li:
                    keyword_count[i]+=1
    res = np.concatenate(([ID],prefixes_count,opcodes_count,keyword_count,registers_count))
    del prefixes_count
    del opcodes_count
    del keyword_count
    del registers_count
    asm_file.close()
    gc.collect()
    return res


def generate_asm_file_features(files,prefixes,opcodes,registers,keywords):
```

```
    '''Generates count asm features'''
    chunk=[]
    for path,file in files:
        chunk.append((file.compute(),path))

    res = [asm_features_util(file,prefixes,opcodes,registers,keywords) for file in tqdm(chunk)]
    del chunk
    gc.collect()
    return res
```

In [10]:

```
prefix = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.
reloc:','.BSS:','.CODE']
opcode = ['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']
keyword = ['.dll','std::',':dword']
register=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
```

In [11]:

```
os.chdir("asmFiles/")
# print(os.getcwd())
```

In [12]:

```
asm_file_objs = jb.Parallel(n_jobs=-2,verbose=2)(jb.delayed(load_asm)(file) for file in (asm_files))
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 11 concurrent workers.
[Parallel(n_jobs=-2)]: Done  19 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-2)]: Done 502 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-2)]: Done 10847 out of 10868 | elapsed:    2.0s remaining:    0.0s
[Parallel(n_jobs=-2)]: Done 10868 out of 10868 | elapsed:    2.0s finished
```

In [13]:

```
%%time
asm_feature_vectors = db.from_sequence(list(zip(asm_files,asm_file_objs)),npartitions=572)
```

```
Wall time: 127 ms
```

In [16]:

```
%%time
asm_feature_vectors = asm_feature_vectors.map_partitions(generate_asm_file_features,prefixes=prefix,opc
odes=opcode,registers=register,keywords=keyword).compute()
print("Done!")
```

```
[########################################] | 100% Completed |  3hr 22min 35.6s
Done!
Wall time: 3h 22min 35s
```

In [21]:

```
asm_feature_vectors = np.vstack(asm_feature_vectors)
asm_feature_vectors.shape
```

Out[21]:

```
(10868, 52)
```

In [25]:

```
asm_cols = ["ID"]
asm_cols+=prefix+opcode+keyword+register
```

```
len(asm_cols)
```

Out[25]:

52

In [34]:

```
# asm_features = pd.DataFrame(asm_feature_vectors,columns=asm_cols)
# asm_features = pd.merge(asm_features, data_size_asm,on='ID', how='left')
# asm_features.to_csv("asm_file_count_features.csv")
# asm_features = pd.read_csv("asm_file_count_features.csv")
asm_features.head()
```

Out[34]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi
0	01azqd4InC7m9JpocGv5	18	22430	0	1158	1366754	0	1794	0	0	...	1891	4371	808	2290	1281
1	01IsoiSMh5gxyDYTl4CB	0	109939	0	616	24568	0	26405	0	0	...	496	1446	260	1090	391
2	01jsnpXSAlgw6aPeDxrU	18	68883	0	304	662	0	1093	0	0	...	4	903	5	547	5
3	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	35	137	18	66	15
4	01SuzwMJEIXsK7A8dQbl	18	10368	0	206	4595	92	0	0	3	...	24	1220	18	1228	24

5 rows × 54 columns

In [41]:

```
# we normalize the data each column
result_asm = normalize(asm_features)
result_asm.head()
```

Out[41]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	
0	01azqd4InC7m9JpocGv5	0.101695	0.032927	0.0	0.006937	0.542847	0.000000	0.000467	0.0	0.000000	...	0.024298	0.03
1	01IsoiSMh5gxyDYTl4CB	0.000000	0.161391	0.0	0.003690	0.009758	0.000000	0.006877	0.0	0.000000	...	0.006373	0.01
2	01jsnpXSAlgw6aPeDxrU	0.101695	0.101121	0.0	0.001821	0.000263	0.000000	0.000285	0.0	0.000000	...	0.000051	0.00
3	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.000000	0.000084	0.0	0.000072	...	0.000450	0.00
4	01SuzwMJEIXsK7A8dQbl	0.101695	0.015220	0.0	0.001234	0.001825	0.012842	0.000000	0.0	0.000072	...	0.000308	0.00

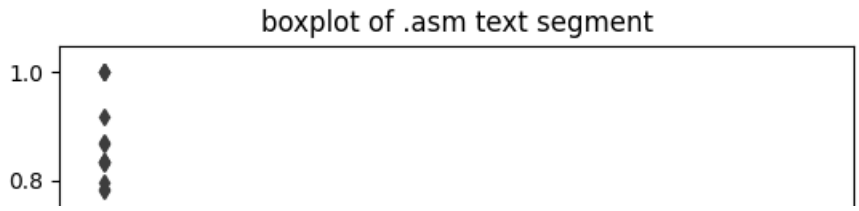
5 rows × 54 columns

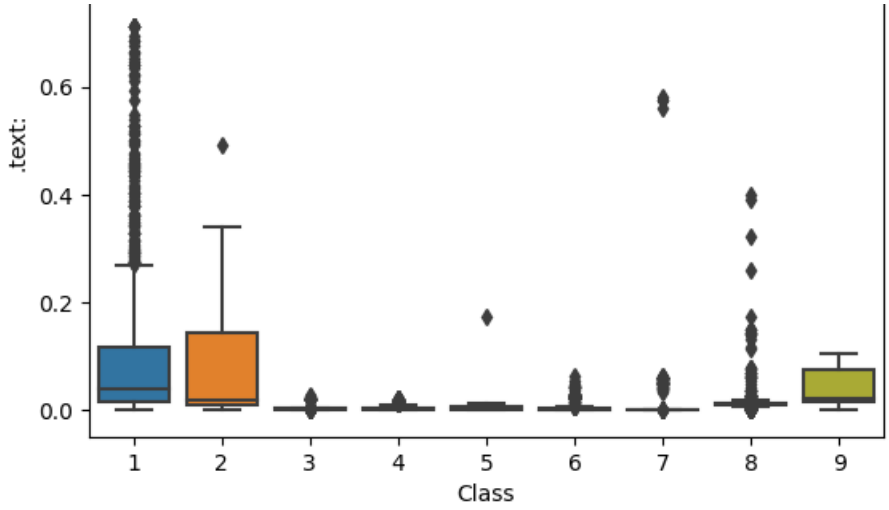
Univariate analysis on asm file features

Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI

In [49]:

```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```

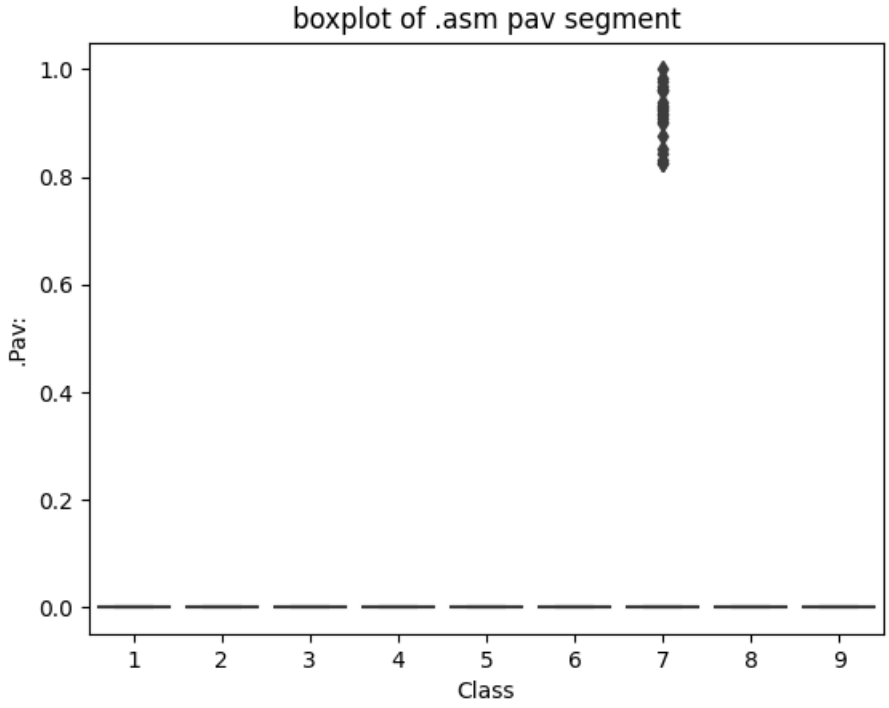




The plot is between Text and class
Class 1,2 and 9 can be easly separated

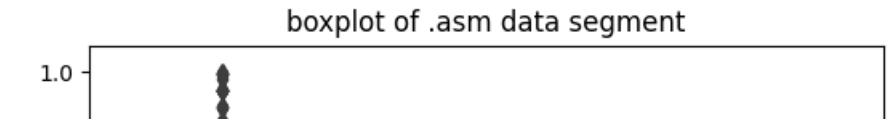
In [48]:

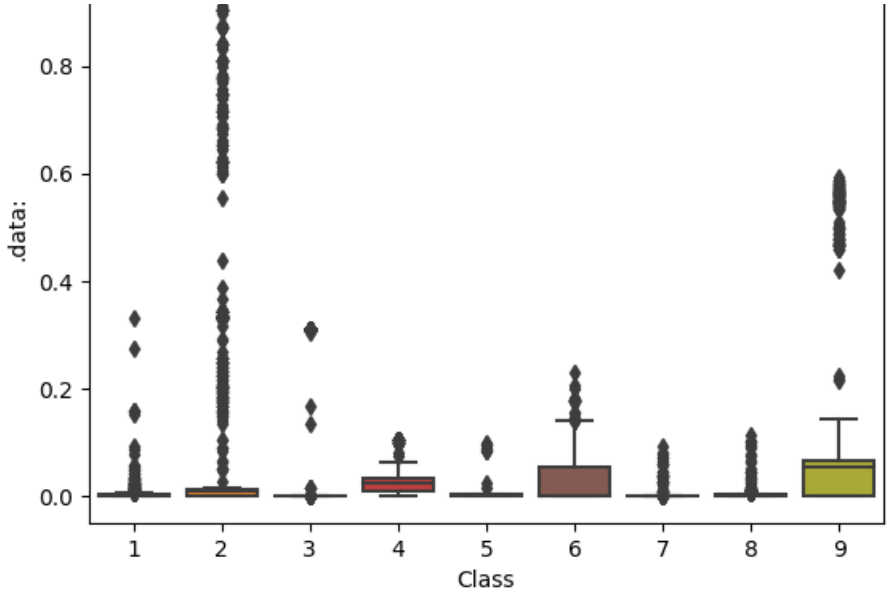
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```



In [50]:

```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```

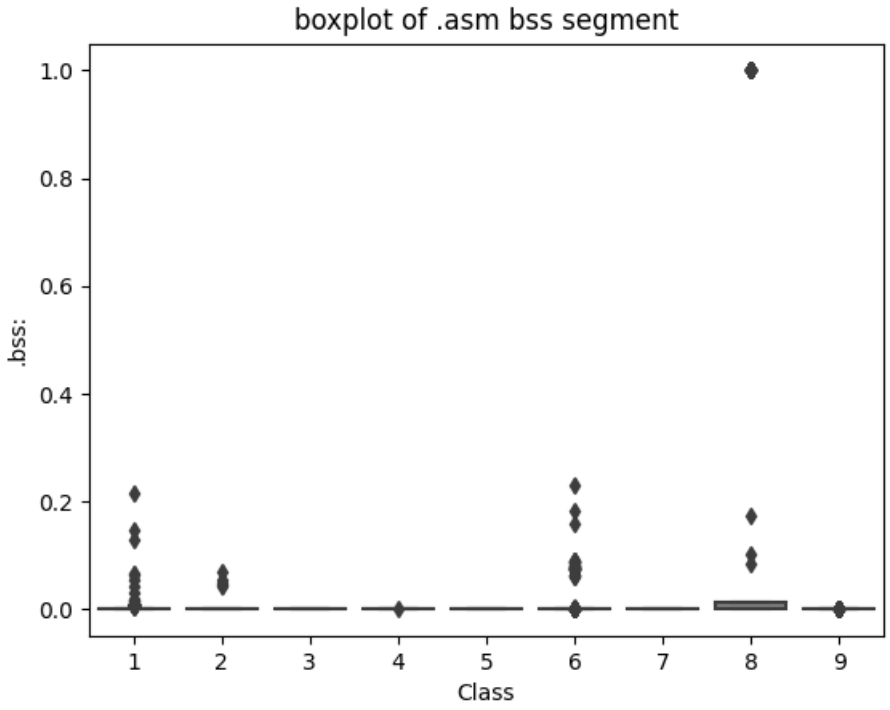




The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [47]:

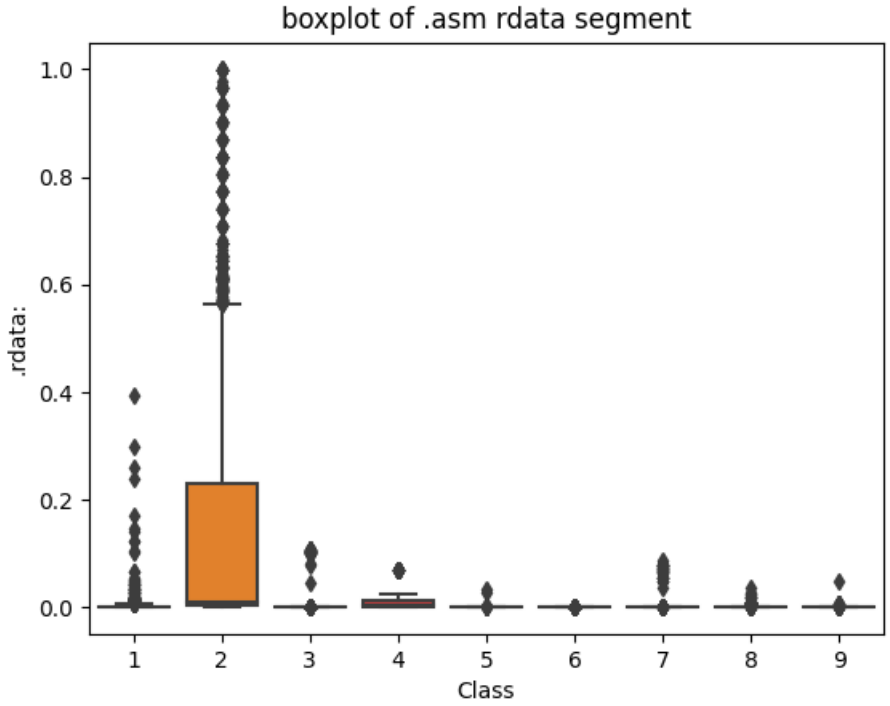
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



plot between bss segment and class label
very less number of files are having bss segment

In [51]:

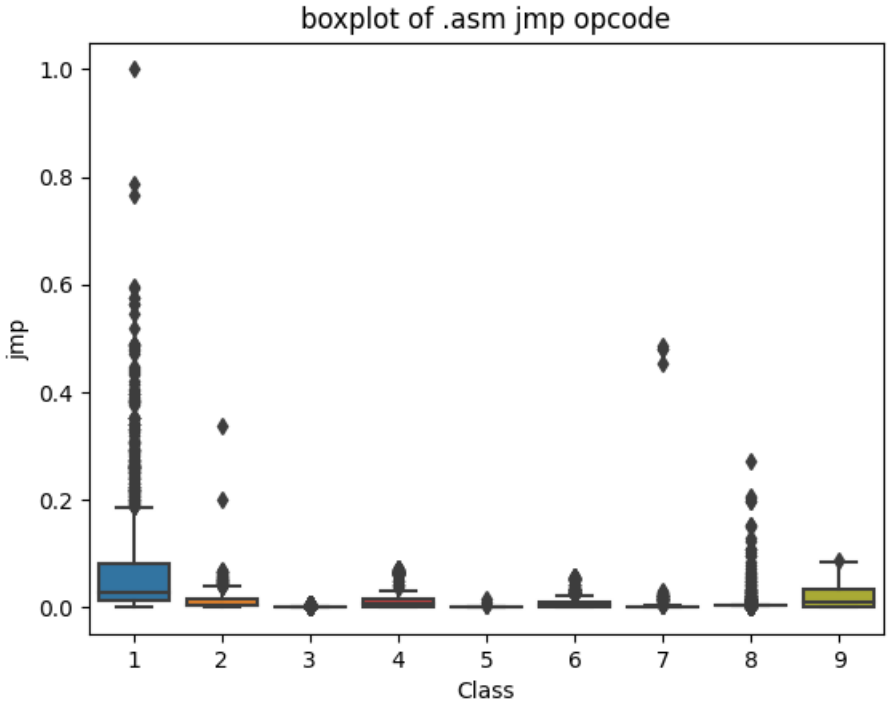
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [52]:

```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```

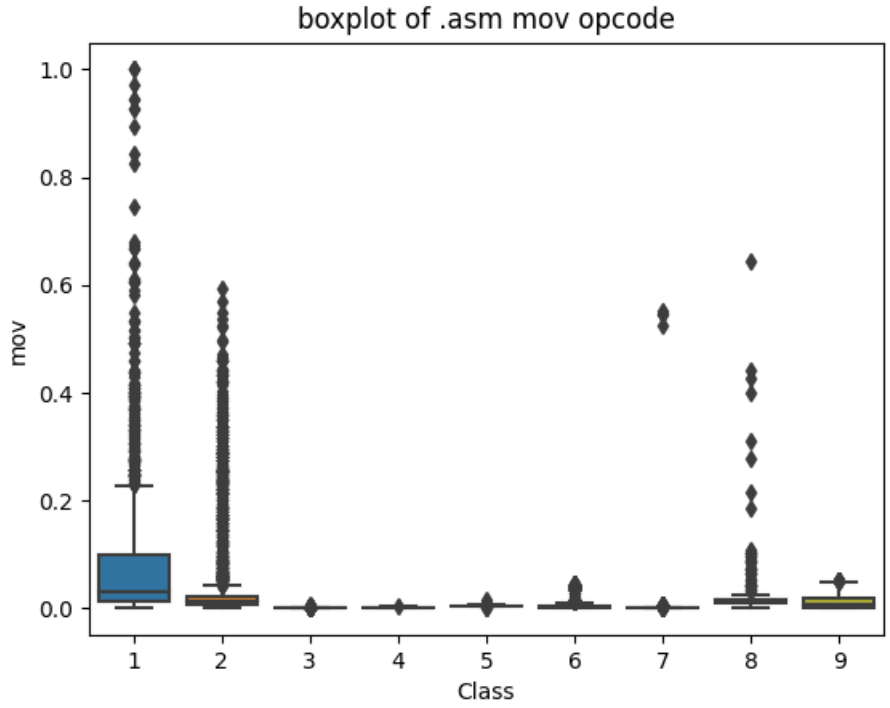


plot between jmp and Class label

plot between jmp and class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [53]:

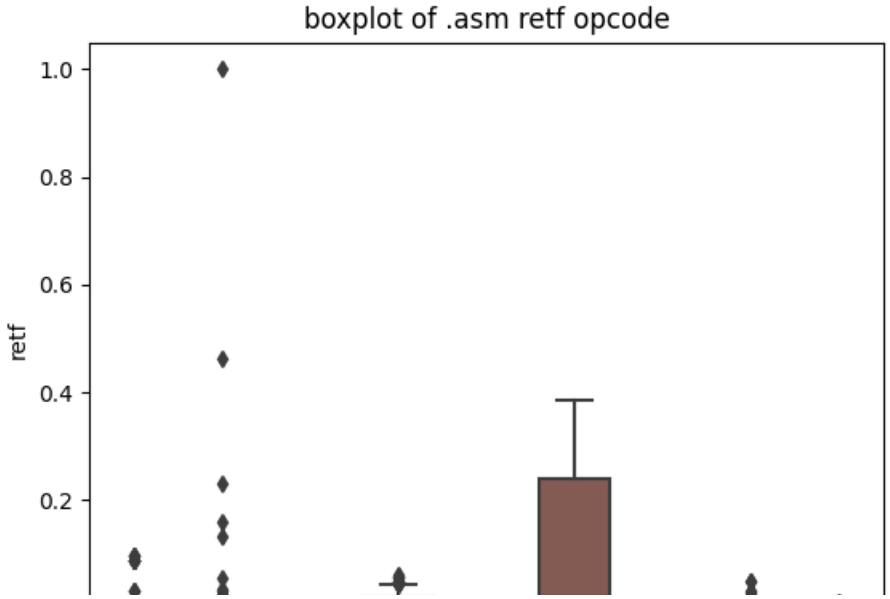
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```

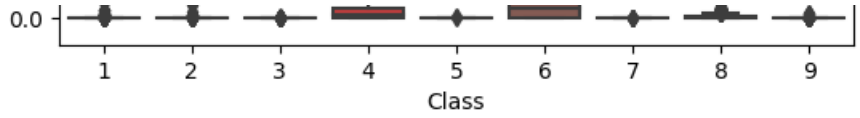


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [54]:

```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```

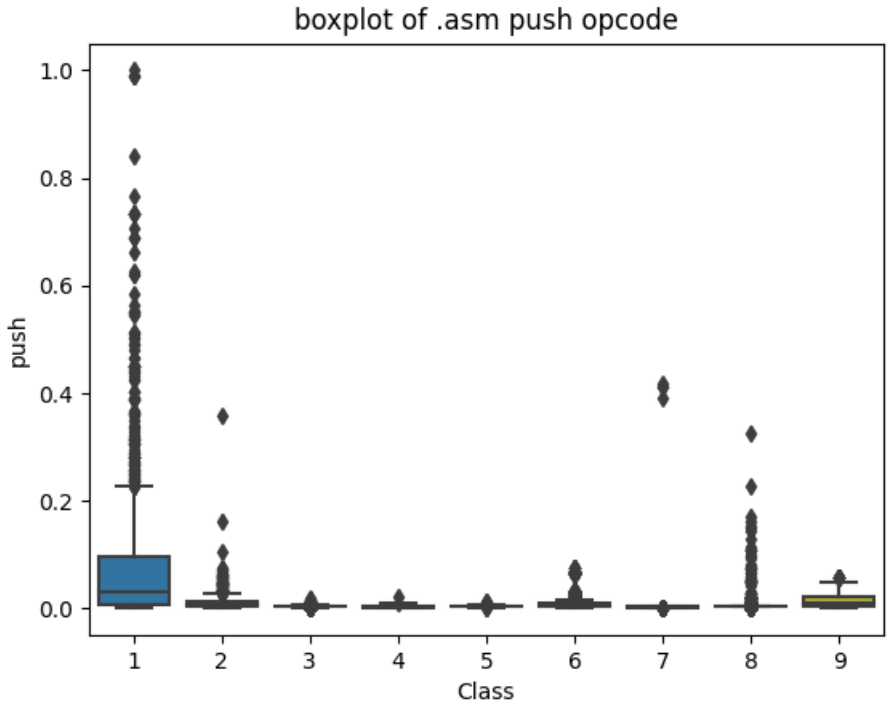




plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [55]:

```
ax = sns.boxplot(x="Class", y="push", data=result_asm)
plt.title("boxplot of .asm push opcode")
plt.show()
```



plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

In [57]:

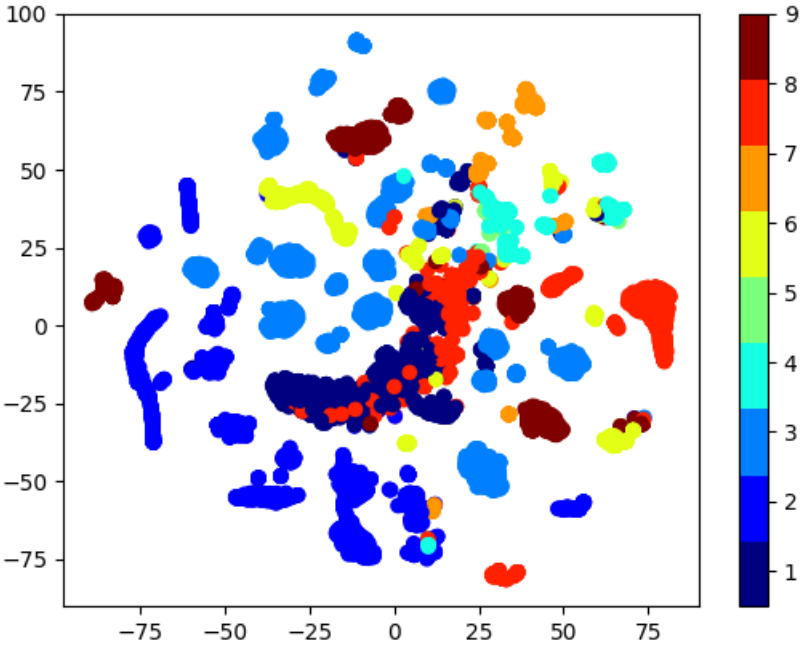
```
data_y = result_asm["Class"]
```

In [58]:

```
#multivariate analysis on byte files
#this is with perplexity 50
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI

xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_asm.drop(['ID','Class'], axis=1).fillna(0))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
```

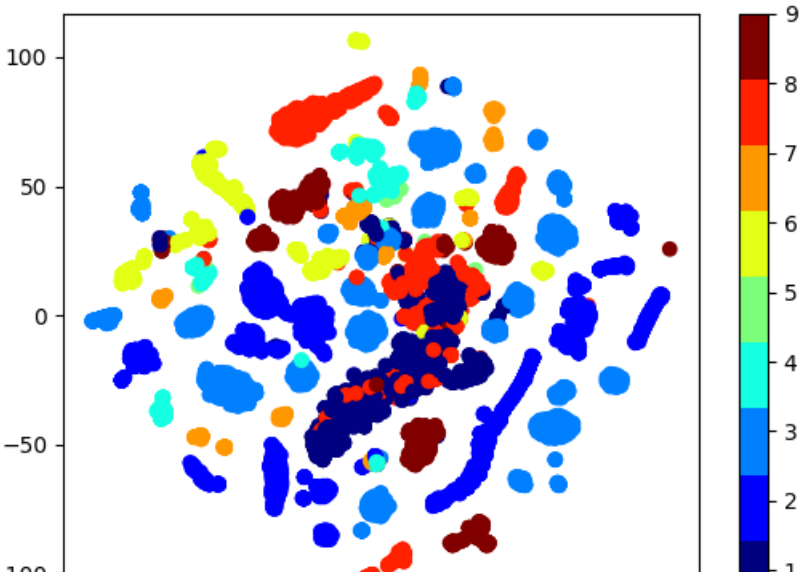
```
plt.show()
```



In [60]:

```
# by univariate analysis on the .asm file features we are getting very negligible information from
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing those fe
atures
# the plot looks very messy
# Ref: MicrosoftMalwareDetection.ipynb provided by Applied AI


xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.CODE','size'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



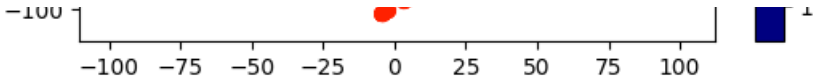


Image Features of .byte and .asm files

In [9]:

```
# Code taken from Ref: https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.ipynb
# Idea implemented by Say_No_to_Overfitting ..
# Research idea proposed in http://vizsec.org/files/2011/Nataraj.pdf
@delayed
def load_file_bin(file):
    return open(file,'rb')

def read_image(file):
#     f = open(filename,'rb')
    ln = os.path.getsize(file.name) # length of file in bytes
    print(ln)
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
    a.fromfile(file,ln-rem)
    file.close()
#     print(type(a))
    g = np.reshape(a,(len(a)//width,width))
    g = np.uint8(g)
    g=g.copy()
    g.resize((1000,))
#     print(g[:50])
    del a
    gc.collect()
    return g

def file_image_util(file):
    asm_file, ID = file
    ID=os.path.basename(ID).split('.')[0]
    image_data = read_image(asm_file)
    return np.append([ID],image_data)


def generate_file_image_features(files):
    '''Generates count asm features'''
    chunk=[]
    for path,file in files:
        chunk.append((file.compute(),path))

    res = [file_image_util(file) for file in tqdm(chunk)]
    del chunk
    gc.collect()
    return res
```

In [10]:

```
column_names = ['ID'] + [("ASM_{:s}".format(str(x))) for x in range(1000)]
```

In [11]:

```
os.chdir("asmFiles/")
# print(os.getcwd())
```

In [12]:

```
asm_file_objs = jb.Parallel(n_jobs=-2,verbose=2)(jb.delayed(load_file_bin)(file) for file in (asm_files))
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 11 concurrent workers.
[Parallel(n_jobs=-2)]: Done  19 tasks      | elapsed:    0.9s
```

```
[Parallel(n_jobs=-2)]: Done 667 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-2)]: Done 10868 out of 10868 | elapsed:    1.8s finished
```

In [13]:

```
%%time
asm_img_feature_vectors = db.from_sequence(list(zip(asm_files,asm_file_objs)),npartitions=209)
```

```
Wall time: 108 ms
```

In [14]:

```
%%time
asm_img_vectors = asm_img_feature_vectors.map_partitions(generate_file_image_features).compute()
os.chdir("../")
print("Done!")
```

```
[########################################] | 100% Completed | 25min 42.2s
Done!
Wall time: 25min 42s
```

In [16]:

```
asm_img_vectors = np.vstack(asm_img_vectors)
asm_img_vectors.shape
```

Out[16]:

```
(10868, 1001)
```

In [19]:

```
# asm_img_features = pd.DataFrame(asm_img_vectors,columns=column_names)
# asm_img_features = pd.merge(asm_img_features, data_size_asm,on='ID', how='left')
# asm_img_features.to_csv("asm_img_features.csv")
# # asm_features = pd.read_csv("asm_file_count_features.csv")
print("Done!!")
```

```
Done!!
```

In [20]:

```
asm_img_features.head()
```

Out[20]:

	ID	ASM_0	ASM_1	ASM_2	ASM_3	ASM_4	ASM_5	ASM_6	ASM_7	ASM_8	...	ASM_992	ASM_993	ASM
0	01azqd4InC7m9JpocGv5	72	69	65	68	69	82	58	48	48	...	120	116	
1	01IsoiSMh5gxyDYTl4CB	46	116	101	120	116	58	48	48	52	...	116	101	
2	01jsnpXSAlgw6aPeDxrU	72	69	65	68	69	82	58	48	48	...	120	116	
3	01kcPWA9K2BOxQeS5Rju	72	69	65	68	69	82	58	49	48	...	69	78	
4	01SuzwMJEIXsK7A8dQbl	72	69	65	68	69	82	58	48	48	...	120	116	

5 rows × 1003 columns

.byte file Image Features

In [14]:

```
os.chdir("byteFiles/")
```

In [42]:

```
column_names = ['ID'] + [("BYTE_{:s}".format(str(x))) for x in range(1000)]
```

In [54]:

```
byte_img_file_objs = jb.Parallel(n_jobs=-2,verbose=2)(jb.delayed(load_file_bin)(file) for file in (hexd
umps))
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 11 concurrent workers.
[Parallel(n_jobs=-2)]: Done  19 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-2)]: Done 667 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-2)]: Done 10868 out of 10868 | elapsed:    1.9s finished
```

In [55]:

```
%%time
byte_img_feature_vectors = db.from_sequence(list(zip(hexdumps,byte_img_file_objs)),npartitions=209)
```

```
Wall time: 61.8 ms
```

In [20]:

```
%%time
byte_img_vectors = byte_img_feature_vectors.map_partitions(generate_file_image_features).compute()
os.chdir("../")
print("Done!")
```

```
[########################################] | 100% Completed | 17min 42.4s
Done!
Wall time: 17min 42s
```

In [24]:

```
byte_img_vectors = np.vstack(byte_img_vectors)
byte_img_vectors.shape
```

Out[24]:

```
(10868, 1001)
```

In [43]:

```
byte_img_features =  pd.DataFrame(byte_img_vectors,columns=column_names)
# byte_img_features.to_csv("byte_img_features.csv")
byte_img_features.head()
```

Out[43]:

	ID	BYTE_0	BYTE_1	BYTE_2	BYTE_3	BYTE_4	BYTE_5	BYTE_6	BYTE_7	BYTE_8	...	BYTE_990	BYTE_9
0	01azqd4InC7m9JpocGv5.txt	69	56	32	48	66	32	48	48	32	...	48	
1	01IsoiSMh5gxyDYTl4CB.txt	67	55	32	48	49	32	50	52	32	...	48	
2	01jsnpXSAlgw6aPeDxrU.txt	67	66	32	67	66	32	67	66	32	...	53	
3	01kcPWA9K2BOxQeS5Rju.txt	54	65	32	70	70	32	54	56	32	...	48	
4	01SuzwMJEIXsK7A8dQbl.txt	65	52	32	65	67	32	52	65	32	...	52	

5 rows × 1001 columns

In [45]:

```
byte_img_features["ID"] = byte_img_features["ID"].str.replace(".txt","")
```

In [48]:

```
# byte_img_features.to_csv("byte_img_features.csv")
```

In [46]:

```
byte_img_features = pd.merge(byte_img_features, data_size_byte,on='ID', how='left')
byte_img_features.head()
```

Out[46]:

	ID	BYTE_0	BYTE_1	BYTE_2	BYTE_3	BYTE_4	BYTE_5	BYTE_6	BYTE_7	BYTE_8	...	BYTE_992	BYTE_993
0	01azqd4InC7m9JpocGv5	69	56	32	48	66	32	48	48	32	...	48	48
1	01IsoiSMh5gxyDYTl4CB	67	55	32	48	49	32	50	52	32	...	48	48
2	01jsnpXSAlgw6aPeDxrU	67	66	32	67	66	32	67	66	32	...	54	56
3	01kcPWA9K2BOxQeS5Rju	54	65	32	70	70	32	54	56	32	...	49	48
4	01SuzwMJEIXsK7A8dQbl	65	52	32	65	67	32	52	65	32	...	54	51

5 rows × 1003 columns

Conclusion on EDA

- We have taken only 52 features from asm files (after reading through many blogs and research papers)
- The univariate analysis was done only on few important features.
- Take-aways
  - 1. Class 3 can be easily separated because of the frequency of segments,opcodes and keywords being less
  - 2. Each feature has its unique importance in separating the Class labels.