Microsoft Malware detection

1.Business/Real-world Problem

1.1. 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

1.2. 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
- 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:nothing, 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(ch
.text:0040100D C7 06 08    BB 42 00                         mov     dword ptr [esi],    offset off_42BB08
.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_42BB08
.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_42BB08
.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 delete(void *)
.text:0040104B 83 C4 04                                add     esp, 4
.text:0040104E
.text:0040104E                              loc_40104E:                     ; CODE XREF: .text:00401043 j
.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 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
```

2.2. Mapping the real-world problem to an ML problem

2.2.1. 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
```

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. 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.

2.3. Train and Test Dataset

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

2.4. Useful blogs, videos and reference papers

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://www.youtube.com/watch?v=VLQTRlLGz5Y

https://github.com/dchad/malware-detection

http://vizsec.org/files/2011/Nataraj.pdf

https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0

" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

```
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
import pickle
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 xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier


#separating byte files and asm files

source = 'train'
destination_1 = 'byteFiles'
destination_2 = 'asmFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a folder with the same name
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

# if we have folder called 'train' (train folder contains both .asm files and .bytes files) we will rename it 'asmFiles'
# for every file that we have in our 'asmFiles' directory we check if it is ending with .bytes, if yes we will move it to
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    data_files = os.listdir(source)
    for file in data_files:
        print(file)
        if (file.endswith("bytes")):
            shutil.move(source+'\\'+file,destination_1)
        if (file.endswith("asm")):
            shutil.move(source+'\\'+file,destination_2)
```

3.1. Distribution of malware classes in whole data set

```
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()
```

3.2. Feature extraction

3.2.1 File size of byte files as a feature

```
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
```

```
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

```
Empty DataFrame
Columns: [ID, size, Class]
Index: []
```

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

```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-e7cc3dc8875f> in <module>()
      1 #boxplot of byte files
```

3.2.3 feature extraction from byte files

```
      2 plt.show()
```

```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        file=file.split('.')[0]
        text_file = open('byteFiles/'+file+".txt", 'w+')
        with open('byteFiles/'+file+".bytes","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+".bytes")
        text_file.close()

files = os.listdir('byteFiles')
filenames2=[]
feature_matrix = np.zeros((len(files),257),dtype=int)
k=0
```

```
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,2
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i, row in enumerate(feature_matrix[k]):
        if i!=len(feature_matrix[k])-1:
            byte_feature_file.write(str(row)+",")
        else:
            byte_feature_file.write(str(row))
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()


byte_features=pd.read_csv("result.csv")
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)


data_size_byte.head(2)
```

```
byte_features_with_size = byte_features.merge(data_size_byte, on='ID')
byte_features_with_size.to_csv("result_with_size.csv")
byte_features_with_size.head(2)
```

```
# 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')):
            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(byte_features_with_size)
```

```
result.head(2)
```

```
data_y = result['Class']
result.head()
```

3.2.4 Multivariate Analysis

```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], 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()
```

```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], 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()
```

Train Test split

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_trai
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
```

```
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_class_distribution.valu


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_class_distribution.values

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
```

```
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_distribution.values[i]/


def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
```

```
print("-"*50, "Confusion matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*50, "Precision matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of columns in precision matrix",B.sum(axis=0))

# representing B in heatmap format
print("-"*50, "Recall matrix"    , "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

```
# we need to generate 9 numbers and the sum of numbers should be 1
```

```
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

4.1.2. K Nearest Neighbour Classification

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
```

```
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClass
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)  Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
```

```
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

4.1.3. Logistic Regression

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.
```

```
#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-------------------------------

alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes_, eps=1e-15))
```

```
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

4.1.4. Random Forest Classifier

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
```

```
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

4.1.5. XgBoost Classification

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)
```

```
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
```

```
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl1.fit(X_train,y_train)

print (random_cfl1.best_params_)


# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, max_depth=3)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

4.2 Modeling with .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 the cores that are present in our com


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
```

4.2.1 Feature extraction from asm files

- To extract the unigram features from the .asm files we need to process ~150GB of data
- **Note: Below two cells will take lot of time (over 48 hours to complete)**
- We will provide you the output file of these two cells, which you can directly use it

```
#intially create five folders
#first
#second
#third
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='first'
folder_2 ='second'
folder_3 ='third'
folder_4 ='fourth'
folder_5 ='fifth'
folder_6 = 'output'
for i in [folder_1,folder_2,folder_3,folder_4,folder_5,folder_6]:
    if not os.path.isdir(i):
        os.makedirs(i)

source='train/'
files = os.listdir('train')
#ID=df['Id'].tolist()
data=range(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'first')
    elif i%5==1:
        shutil.move(source+files[data[i]],'second')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'third')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fifth')
```

```
#http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html

def firstprocess():
    #The prefixes tells about the segments that are present in the asm files
    #There are 450 segments(approx) present in all asm files.
    #this prefixes are best segments that gives us best values.
    #https://en.wikipedia.org/wiki/Data_segment

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cm
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
        with codecs.open('first/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                # https://www.tutorialspoint.com/python3/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
```

```
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


#same as above
def secondprocess():
    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', 'cm
keywords = ['.dll','std::',':dword']
registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
file1=open("output\mediumasmfile.txt","w+")
files = os.listdir('second')
for f in files:
    prefixescount=np.zeros(len(prefixes),dtype=int)
    opcodescount=np.zeros(len(opcodes),dtype=int)
    keywordcount=np.zeros(len(keywords),dtype=int)
    registerscount=np.zeros(len(registers),dtype=int)
    features=[]
    f2=f.split('.')[0]
    file1.write(f2+",")
    opcodefile.write(f2+" ")
    with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:
        for lines in fli:
            line=lines.rstrip().split()
            l=line[0]
            for i in range(len(prefixes)):
                if prefixes[i] in line[0]:
                    prefixescount[i]+=1
            line=line[1:]
            for i in range(len(opcodes)):
                if any(opcodes[i]==li for li in line):
                    features.append(opcodes[i])
                    opcodescount[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):
                        registerscount[i]+=1
            for i in range(len(keywords)):
                for li in line:
                    if keywords[i] in li:
                        keywordcount[i]+=1
    for prefix in prefixescount:
        file1.write(str(prefix)+",")
    for opcode in opcodescount:
```

```
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()

# same as smallprocess() functions
def thirdprocess():
    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', 'cm
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('third')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('third/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[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):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
            for prefix in prefixescount:
                file1.write(str(prefix)+",")
            for opcode in opcodescount:
                file1.write(str(opcode)+",")
            for register in registerscount:
                file1.write(str(register)+",")
            for key in keywordcount:
                file1.write(str(key)+",")
            file1.write("\n")
        file1.close()


    def fourthprocess():
        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', 'cm
        keywords = ['.dll','std::',':dword']
        registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
        file1=open("output\hugeasmfile.txt","w+")
        files = os.listdir('fourth/')
        for f in files:
            prefixescount=np.zeros(len(prefixes),dtype=int)
            opcodescount=np.zeros(len(opcodes),dtype=int)
            keywordcount=np.zeros(len(keywords),dtype=int)
            registerscount=np.zeros(len(registers),dtype=int)
            features=[]
            f2=f.split('.')[0]
            file1.write(f2+",")
            opcodefile.write(f2+" ")
            with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as fli:
```

```
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[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):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fifthprocess():
    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', 'cm
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
```

```
files = os.listdir('fifth/')
for f in files:
    prefixescount=np.zeros(len(prefixes),dtype=int)
    opcodescount=np.zeros(len(opcodes),dtype=int)
    keywordcount=np.zeros(len(keywords),dtype=int)
    registerscount=np.zeros(len(registers),dtype=int)
    features=[]
    f2=f.split('.')[0]
    file1.write(f2+",")
    opcodefile.write(f2+" ")
    with codecs.open('fifth/'+f,encoding='cp1252',errors ='replace') as fli:
        for lines in fli:
            line=lines.rstrip().split()
            l=line[0]
            for i in range(len(prefixes)):
                if prefixes[i] in line[0]:
                    prefixescount[i]+=1
            line=line[1:]
            for i in range(len(opcodes)):
                if any(opcodes[i]==li for li in line):
                    features.append(opcodes[i])
                    opcodescount[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):
                        registerscount[i]+=1
            for i in range(len(keywords)):
                for li in line:
                    if keywords[i] in li:
                        keywordcount[i]+=1
    for prefix in prefixescount:
        file1.write(str(prefix)+",")
    for opcode in opcodescount:
        file1.write(str(opcode)+",")
    for register in registerscount:
        file1.write(str(register)+",")
    for key in keywordcount:
```

```
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present System
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
    p3=Process(target=thirdprocess)
    p4=Process(target=fourthprocess)
    p5=Process(target=fifthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

if __name__=="__main__":
    main()


# asmoutputfile.csv(output genarated from the above two cells) will contain all the extracted features from .asm files
# this file will be uploaded in the drive, you can directly use this
dfasm=pd.read_csv("asmoutputfile.csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='left')
result_asm.head()
```

4.2.1.1 Files sizes of each .asm file

```
#file sizes of byte files

files=os.listdir('asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

4.2.1.2 Distribution of .asm file sizes

```
#boxplot of asm files
ax = sns.boxplot(x="Class", y="size", data=asm_size_byte)
```

```
plt.title("boxplot of .bytes file sizes")
```

```
# add the file size feature to previous extracted features
print(result_asm.shape)
print(asm_size_byte.shape)
result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='ID', how='left')
result_asm.head()
```

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

4.2.2 Univariate analysis on asm file features

```
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
```

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

```
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
```

```
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
```

```
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
```

```
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
Class 1 is having frequency of 2000 approx in 75 perentile of files
```

```
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
```

```
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.
```

```
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

```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic-neighbourhood-embeddingt-sne-part-
```

```
#multivariate analysis on byte files
#this is with perplexity 50
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()


# 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 features
# the plot looks very messy

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()
```

```
TSNE for asm data with perplexity 50
```

4.2.3 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.

4.3 Train and test split

```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y_train_asm,test_size=0.20)
```

```
print( X_cv_asm.isnull().all())
```

4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
```

```
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClass
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)  Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',log_loss(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',log_loss(y_cv_asm, predict_y))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',log_loss(y_test_asm, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

4.4.2 Logistic Regression

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.
```

```
#--------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------


alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
```

```
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

4.4.3 Random Forest Classifier

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])  Fit the SVM model according to the given training data.
# predict(X)  Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
```

```
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

4.4.4 XgBoost Classifier

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data
```

```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
```

```
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_asm, predict_y))
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_asm, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

4.4.5 Xgboost Classifier with best hyperparameters

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_asm,y_train_asm)
```

```
print (random_cfl.best_params_)


# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5,max_depth=3)
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

predict_y = c_cfl.predict_proba(X_train_asm)
print ('train loss',log_loss(y_train_asm, predict_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict_y))
```

4.5. Machine Learning models on features of both .asm and .bytes files

4.5.1. Merging both asm and byte file features

```
result.head()
```

```
result_asm.head()
```

```
print(result.shape)
print(result_asm.shape)
```

```
result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

4.5.2. Multivariate Analysis on final fearures

```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x, axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```

4.5.3. Train and Test split

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
```

```
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

4.5.4. Random Forest Classifier on final features

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=r_cfl.classes_, eps=1e-15))
```

```
for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_y))
```

4.5.5. XgBoost Classifier on final features

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
```

```
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
```

```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=3000,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_y))
```

4.5.5. XgBoost Classifier on final features with best hyper parameters using Random search

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_merge, y_train_merge)


print (random_cfl.best_params_)
```

```
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.3,subsample=1,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video) and include pixel intensity features to improve the logloss

```
1. you need to donwload the train from kaggle, which is of size ~17GB, after extracting it will occupy ~128GB data your dirve

2. if you are having computation power limitations, you can try using google colab, with GPU option enabled (you can search for how

3. To Extract the .7z file in google cloud, once after you upload the file into server, in your ipython notebook create a new cell
  a. !sudo apt-get install p7zip
  b. !7z x file_name.7z -o path/where/you/want/to/extract

https://askubuntu.com/a/341637
```

```
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
import pickle
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 xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHT
```

```
--2022-06-06 13:31:27--  https://storage.googleapis.com/kagglesdsdata/competitions/4117/46665/train.7z?GoogleAccessId=web-data@
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.193.128, 172.217.204.128, 172.253.123.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.193.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18810691091 (18G) [application/x-7z-compressed]
Saving to: ‘train.7z’

train.7z            100%[===================>]  17.52G   185MB/s    in 2m 26s

2022-06-06 13:33:53 (123 MB/s) - ‘train.7z’ saved [18810691091/18810691091]
```

```
data=!7z l train.7z
print(len(data))
print(type(data))
```

```
21759
<class 'IPython.utils.text.SList'>
```

```
byte_file_list=[]
asm_file_list=[]
for i in data:
  if (i.endswith("bytes")):
    byte_file_list.append(i)
```

```
  elif (i.endswith("asm")):
    asm_file_list.append(i)

print('Number of Byte files',len(byte_file_list))
print('Number of ASM files',len(asm_file_list))
print(asm_file_list[0])
```

```
Number of Byte files 10868
Number of ASM files 10868
2015-01-29 05:00:00 ....A     58961309    174379443  train/01azqd4InC7m9JpocGv5.asm
```

```
byte_file_name=[]
for i in range(0,len(byte_file_list)):
  byte_file_name.append(byte_file_list[i].split()[-1].replace('train/', ''))
```

```
asm_file_name=[]
for i in range(0,len(asm_file_list)):
  asm_file_name.append(asm_file_list[i].split()[-1].replace('train/', ''))
```

```
byte_file_name[:3]
```

```
['01azqd4InC7m9JpocGv5.bytes',
 '01IsoiSMh5gxyDYTl4CB.bytes',
 '01jsnpXSAlgw6aPeDxrU.bytes']
```

```
asm_file_name[:3]
```

```
['01azqd4InC7m9JpocGv5.asm',
 '01IsoiSMh5gxyDYTl4CB.asm',
 '01jsnpXSAlgw6aPeDxrU.asm']
```

```
import os
# Getting the current working directory
os.getcwd()
```

```
'/content'
```

```
os.listdir()
```

```
['.config', 'train.7z', 'sample_data']
```

```
from datetime import datetime
start=datetime.now()

!7z e train.7z -o/content/byteFiles *.bytes -r

end=datetime.now()
difference=end-start
print('Time taken to extract all the byte files',difference)
```

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Xeon(R) CPU @ 2.20GHz (406F0),ASM,AES-NI)

Scanning the drive for archives:
1 file, 18810691091 bytes (18 GiB)

Extracting archive: train.7z
--
Path = train.7z
Type = 7z
Physical Size = 18810691091
Headers Size = 339764
Method = LZMA:24
Solid = +
Blocks = 94

Everything is Ok

Files: 10931
Size:       51652298914
```

```
Compressed: 18810691091
Time taken to extract all the byte files 0:14:51.110295
```

```
source = 'train'
destination_1 = 'byteFiles'
destination_2 = 'asmFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a folder with the same name
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

# if we have folder called 'train' (train folder contains both .asm files and .bytes files) we will rename it 'asmFiles'
# for every file that we have in our 'asmFiles' directory we check if it is ending with .bytes, if yes we will move it to
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    data_files = os.listdir(source)
    for file in data_files:
        print(file)
        if (file.endswith("bytes")):
            shutil.move(source+'\\'+file,destination_1)
        if (file.endswith("asm")):
            shutil.move(source+'\\'+file,destination_2)


from tqdm import tqdm


result = pd.read_csv('result.csv')


vocab=result.columns[1:]
byte_vocab =[]
for i in vocab:
  for j in range(len(vocab)):
    byte_vocab.append(i+' '+vocab[j])
```

```
print(len(byte_vocab))
print(byte_vocab[:5])
```

```
66049
['0 0', '0 1', '0 2', '0 3', '0 4']
```

```
from scipy.sparse import csr_matrix
from sklearn.feature_extraction.text import CountVectorizer
import csv
vectorizer = CountVectorizer(tokenizer=lambda x: x.split(),lowercase=False,ngram_range=(2,2),vocabulary=byte_vocab)
file_lists=os.listdir('byteFiles')
features=["ID"]+vectorizer.get_feature_names()
byte_bigram_df=pd.DataFrame(columns=features)
```

```
with open('byte_bigram_df.csv','w') as byte_bigram_df:
  byte_bigram_df.write(','.join(map(str, features)))
  byte_bigram_df.write('\n')
  for _, file in tqdm(enumerate(file_lists)):
    file_id=file.split(".")[0]
    file = open('byteFiles/' + file)
    corpus=[file.read().replace('\n', ' ').lower()]
    bigrams=vectorizer.transform(corpus)
    row=csr_matrix(bigrams).toarray()
    byte_bigram_df.write(','.join(map(str, [file_id]+list(row[0]))))
    byte_bigram_df.write('\n')
    file.close()
```

```
10868it [3:15:20,  1.08s/it]
```

```
df_2 =pd.read_csv('trainLabels.csv')
df_2.head()
```

	Id	Class
0	01kcPWA9K2BOxQeS5Rju	1
1	04EjIdbPV5e1XroFOpiN	1
2	05EeG39MTRrI6VY21DPd	1
3	05rJTUWYAKNegBk2wE8X	1

```
labels = df_2['Class']
```

```
!unzip byte_bigram_df.zip
```

```
Archive:  byte_bigram_df.zip
  inflating: byte_bigram_df.csv
```

```
import dask.dataframe as dd
X=dd.read_csv('byte_bigram_df.csv',sample =256000000)
y=labels
```

```
X.head(1)
```

	ID	00	01	02	03	04	05	06	07	08	...	??f7	??f8	??f9	??fa	??fb	??fc	??fd	??fe	??ff	?? ??
0	Hmne203D6GVOJkqhsi8y	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	12869

1 rows × 66050 columns

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

kbest_object=SelectKBest(score_func=chi2, k=1000)
best_features=kbest_object.fit(X.drop("ID", axis=1),y)
df_scores=pd.DataFrame(best_features.scores_)
df_columns=pd.DataFrame(X.columns)
```

```
feature_score_df=pd.concat([df_columns,df_scores],axis=1)
feature_score_df.columns=["Feature_Name","Feature_Score"]

#Let's look at the top 1000 features along with their scores + Save the feature score DF
feature_score_df=feature_score_df.nlargest(1000,"Feature_Score")
feature_score_df.to_csv("byte_bigram_df1.csv", index=None)
feature_score_df.head(5)
```

	Feature_Name	Feature_Score
66048	?? ff	1.110606e+07
9947	26 b4	3.743977e+06
9997	26 e6	3.609365e+06
9966	26 c7	3.510049e+06
46555	b5 25	2.993972e+06

```
feature_score_df = pd.read_csv('byte_bigram_df1.csv')
feature_score_df.head(5)
```

	Feature_Name	Feature_Score
0	?? ff	1.110606e+07
1	26 b4	3.743977e+06
2	26 e6	3.609365e+06
3	26 c7	3.510049e+06
4	b5 25	2.993972e+06

```
top_1000_feats=list(feature_score_df["Feature_Name"])

#Get the dataframe containing the top 1000 features
```

```
reduced_features=dd.concat([X["ID"],X[top_1000_feats]], axis=1)

#Save the dataframes containing the top 1000 features
reduced_features.to_csv("features/final_features/temp/top1000_byte_bigram_df*.csv",index=None)
#del(X, feature_score_df, reduced_features)
```

```
['/content/features/final_features/temp/top1000_byte_bigram_df00.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df01.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df02.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df03.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df04.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df05.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df06.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df07.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df08.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df09.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df10.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df11.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df12.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df13.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df14.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df15.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df16.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df17.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df18.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df19.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df20.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df21.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df22.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df23.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df24.csv',
 '/content/features/final_features/temp/top1000_byte_bigram_df25.csv']
```

```
del(X, feature_score_df, reduced_features)
```

```
file_lists=os.listdir('features/final_features/temp/')
final_byte_bigram_df=pd.read_csv('features/final_features/temp/'+file_lists[0])
```

```
for i in range(1,len(file_lists)):
    df = pd.read_csv('features/final_features/temp/'+file_lists[i])
    final_byte_bigram_df=pd.concat([final_byte_bigram_df,df], axis=0)

final_byte_bigram_df.to_csv("features/final_features/top1000_byte_bigram_df.csv",index=None)
del(final_byte_bigram_df, file_lists)


top1000_byte_bigram_df = pd.read_csv("features/final_features/top1000_byte_bigram_df.csv").drop(columns=["ID"])


top1000_byte_bigram_df.head(2)
```

	?? ff	26 b4	26 e6	26 c7	b5 25	e7 25	26 93	26 94	ba 0a	c8 25	...	4e 45	47 3d	f3 20	95 94	a9 67	0a 5e	81 20	f4 f3	34 71	87 2f
0	0	2	0	2	2	0	2	0	1	3	...	2	1	2	1	1	0	1	0	0	3
1	0	0	1	2	1	1	0	1	2	2	...	0	0	2	1	0	1	0	4	0	1

2 rows × 1000 columns

Merging the result file and asm output file

```
result = pd.read_csv('result_with_size.csv')
print('shape of the dataframe is : ',result.shape)
result.head()
```

```
shape of the dataframe is :  (10868, 261)
```

Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9	fa	fb	fc	fd	fe

```
asm_output = pd.read_csv('asmoutputfile.csv')
print("Shape of the asm output file : ",asm_output.shape)
asm_output.head()
```

```
Shape of the asm output file :  (10868, 52)
```

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	:dword	edx	esi	eax
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	137	18	66	15
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	130	18	29	48
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	84	13	42	10
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	25	6	8	14
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	18	12	9	18

5 rows × 52 columns

```
final_df_1 = pd.merge(result,asm_output,on='ID', how='left')
final_df_1 = pd.merge(final_df_1,top_1000_byte_bigram,on = 'ID',how = 'left')
print('shape of the dataframe is : ',final_df_1.shape)
```

```
shape of the dataframe is :  (10868, 1312)
```

Adding Top 1000 Byte Bigram Features

```
top_1000_byte_bigram = pd.read_csv('top1000_byte_bigram_df.csv')
print('shape of the dataframe is : ',top_1000_byte_bigram.shape)
top_1000_byte_bigram.head()
```

```
shape of the dataframe is :  (10868, 1001)
```

	ID	?? ff	26 b4	26 e6	26 c7	b5 25	e7 25	26 93	26 94	ba 0a	...	4e 45	47 3d	f3 20	95 94	a9 67	0a 5e	81 20	f4 f3	34 71	87 2f
0	LCrZhlaY3MnDA1GKwtvT	0	2	0	2	2	0	2	0	1	...	2	1	2	1	1	0	1	0	0	3
1	aRi6UqPbCVQs1gLWmF4e	0	0	1	2	1	1	0	1	2	...	0	0	2	1	0	1	0	4	0	1
2	crKe1xWtHpsz20LdiVMS	0	8	12	12	11	13	11	11	12	...	9	9	13	8	12	3	6	13	9	10
3	8f34pWz5MiSc2nDtlOPN	0	14	18	24	20	20	16	30	10	...	26	34	26	14	16	32	20	20	18	24
4	FBrgMHOAi6txEZLU5ezT	0	16	14	10	5	10	12	7	16	...	13	13	13	10	8	10	6	9	9	4

5 rows × 1001 columns

```
result_y = final_df_1['Class']
result_x = final_df_1.drop(['ID','rtn','.BSS:','.CODE','Class','Unnamed: 0'], axis=1)
print("shape of result_y : ",result_y.shape)
print('Shape of result_x : ',result_x.shape)
```

```
shape of result_y :  (10868,)
Shape of result_x :  (10868, 1306)
```

```
result_x.head()
```

	0	1	2	3	4	5	6	7	8	9	...	4e	47	f3	95	a9	0a	81	f4	34	87
1	39755	8337	7249	7186	8663	6844	8420	7589	9291	358	...	2	1	0	6	0	0	0	1	1	0

Train and Test Split

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

```
print('Shape of X_train : ',X_train_merge.shape)
print('Shape of cv : ',X_cv_merge.shape)
print('Shape of X_test : ',X_test_merge.shape)
```

```
Shape of X_train :  (6955, 1306)
Shape of cv :  (1739, 1306)
Shape of X_test :  (2174, 1306)
```

Distribution of Each Class

```
train_class_distribution = y_train_merge.value_counts().sort_index()
test_class_distribution = y_test_merge.value_counts().sort_index()
cv_class_distribution = y_cv_merge.value_counts().sort_index()

my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
```

```
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_class_distribution.valu


print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_class_distribution.values

print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_distribution.values[i]/
```

```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
--------------------------------------------------------------------------------
```

Distribution of yi in test data

```
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
--------------------------------------------------------------------------------
```

```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
```

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
```

```
# C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
# C.sum(axix =1) = [[3, 7]]
# ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
#                           [2/3, 4/7]]

# ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
#                           [3/7, 4/7]]
# sum of row elements = 1

B =(C/C.sum(axis=0))
#divid each element of the confusion matrix with the sum of elements in that row
# C = [[1, 2],
#     [3, 4]]
# C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
# C.sum(axix =0) = [[4, 6]]
# (C/C.sum(axis=0)) = [[1/4, 2/6],
#                      [3/4, 4/6]]

labels = [1,2,3,4,5,6,7,8,9]
cmap=sns.light_palette("green")
# representing A in heatmap format
print("-"*50, "Confusion matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*50, "Precision matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of columns in precision matrix",B.sum(axis=0))

# representing B in heatmap format
```

```
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

Machine Learning Models

Random Forest: Hyperparameter Tuning

```
from sklearn.ensemble import RandomForestClassifier
alpha = [10,50,100,500,1000,2000,3000]
cv_error = []

for i in range(len(alpha)):
    rf_clf = RandomForestClassifier(n_estimators=alpha[i],random_state= 42,n_jobs=-1)
    rf_clf.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(rf_clf,method ='sigmoid')
    sig_clf.fit(X_train_merge,y_train_merge)
    y_pred = sig_clf.predict_proba(X_cv_merge)
    loss = log_loss(y_cv_merge,y_pred, labels = rf_clf.classes_ ,  eps=1e-15)
    cv_error.append(loss)

for j in range(len(cv_error)):
    print('log_loss for c = ',alpha[j],'is',cv_error[j])
```

```
 log_loss for c =  10 is 0.06131539643837644
 log_loss for c =  50 is 0.053437534053347197
 log_loss for c =  100 is 0.05213926777133298
 log_loss for c =  500 is 0.05182692930878205
 log_loss for c =  1000 is 0.05177058830001596
 log_loss for c =  2000 is 0.0517810215458865
 log_loss for c =  3000 is 0.05167788427311402
```

```
best_alpha = np.argmin(cv_error)
plt.plot(alpha,cv_error,label = 'logloss')
plt.legend()
plt.grid()
plt.title('Cross valudation error for each epoch')
plt.xlabel("Alpha Values")
plt.ylabel("Error measure")
plt.show()

print('Best aplha value is : ',alpha[best_alpha])
```

```
from sklearn.model_selection import RandomizedSearchCV
model = RandomForestClassifier()
parameters= {
    'n_estimators':[10,50,100,500,1000,2000,3000],
    'max_depth' : [5, 10, 20, 30, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
clf_rs= RandomizedSearchCV(model,param_distributions=parameters,cv=10,scoring= 'neg_log_loss',n_jobs=-1,return_train_score=True)
clf_rs.fit(X_train_merge,y_train_merge)
```

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_job...
                                                    oob_score=False,
                                                    random_state=None,
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [5, 10, 20, 30, 50],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [10, 50, 100, 500, 1000,
                                                         2000, 3000]},
```

```
                         pre_dispatch='2*n_jobs', random_state=None, refit=True,
                         return_train_score=True, scoring='neg_log_loss', verbose=0)
```

```
clf_rs.best_params_
```

```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_depth': 20}
```

```
clf = RandomForestClassifier(n_estimators= 1000,min_samples_split= 2,min_samples_leaf= 1,max_depth= 20,random_state=42,n_jobs = -1)
clf.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(clf)
sig_clf.fit(X_train_merge,y_train_merge)
```

```
CalibratedClassifierCV(base_estimator=RandomForestClassifier(bootstrap=True,
                                                             ccp_alpha=0.0,
                                                             class_weight=None,
                                                             criterion='gini',
                                                             max_depth=20,
                                                             max_features='auto',
                                                             max_leaf_nodes=None,
                                                             max_samples=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=2,
                                                             min_weight_fraction_leaf=0.0,
                                                             n_estimators=1000,
                                                             n_jobs=-1,
                                                             oob_score=False,
                                                             random_state=42,
                                                             verbose=0,
                                                             warm_start=False),
                       cv=None, method='sigmoid')
```

```
y_pred = sig_clf.predict_proba(X_train_merge)
```

```
print('For values of best alpha = 1000 | The train log loss is:',log_loss(y_train_merge, y_pred))
y_pred = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha =  1000 | The CV log loss is:',log_loss(y_cv_merge, y_pred))
y_pred = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha =  1000 | The Test log loss is:',log_loss(y_test_merge, y_pred))
```

```
For values of best alpha = 1000 | The train log loss is: 0.01892213385998057
For values of best alpha =  1000 | The CV log loss is: 0.04186850920356656
For values of best alpha =  1000 | The Test log loss is: 0.053717791869466264
```

```
plot_confusion_matrix(y_test_merge, sig_clf.predict(X_test_merge))
```

```
-------------------------------------------------- Confusion matrix --------------------------------------------------
```

```
-------------------------------------------------- Precision matrix --------------------------------------------------
```

```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
---------------------------------------------------- Recall matrix ----------------------------------------------------
```

XGBoost Classifier : Hyperparameter Tuning

```
from xgboost import XGBClassifier
from sklearn.calibration import CalibratedClassifierCV
alpha = [10,50,100,500,1000,2000,3000]
cv_error = []

for i in range(len(alpha)):
    xg_clf = XGBClassifier(n_estimators=alpha[i],nthread = -1)
    xg_clf.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(base_estimator=xg_clf,method="sigmoid")
    sig_clf.fit(X_train_merge,y_train_merge)
    y_pred = sig_clf.predict_proba(X_cv_merge)
    loss = log_loss(y_cv_merge,y_pred,labels=xg_clf.classes_ ,eps=1e-15)
    cv_error.append(loss)

for j in range(len(cv_error)):
    print('log_loss for c = ',alpha[j],'is',cv_error[j])
```

```
log_loss for c =  10 is 0.06573636947232045
log_loss for c =  50 is 0.05111404662515569
log_loss for c =  100 is 0.051277661288549596
log_loss for c =  500 is 0.049201285323898995
log_loss for c =  1000 is 0.04920108519458488
log_loss for c =  2000 is 0.04920092887808062
log_loss for c =  3000 is 0.04920206000303608
```

```
best_alpha = np.argmin(cv_error)
plt.plot(alpha,cv_error,label = 'logloss')
```

```
plt.legend()
plt.grid()
plt.title('Cross valudation error for each epoch')
plt.xlabel("Alpha Values")
plt.ylabel("Error measure")
plt.show()

print('Best aplha value is : ',alpha[best_alpha])
```

```
Best aplha value is :  2000
```

```
model = XGBClassifier()
```

```
prams={
    'n_estimators':[10,50,100,500,1000,2000,3000],
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
clf_rs = RandomizedSearchCV(model,param_distributions=prams,cv=10,scoring= 'neg_log_loss',n_jobs=-1,return_train_score=True)
clf_rs.fit(X_train_merge,y_train_merge)
```

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None,...
                                           verbosity=None),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [3, 5, 10],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='neg_log_loss', verbose=0)
```

```
clf_rs.best_params_
```

```
{'subsample': 0.5,
 'n_estimators': 2000,
 'max_depth': 3,
```

```
         'learning_rate': 0.03,
         'colsample_bytree': 0.5}
```

```
xg_clf = XGBClassifier(subsample= 0.5,n_estimators= 2000,max_depth= 3,learning_rate= 0.03,colsample_bytree= 0.5,random_state= 42,nthr
xg_clf.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(xg_clf,method ='sigmoid')
sig_clf.fit(X_train_merge,y_train_merge)
```

```
     CalibratedClassifierCV(base_estimator=XGBClassifier(base_score=0.5,
                                                         booster=None,
                                                         colsample_bylevel=1,
                                                         colsample_bynode=1,
                                                         colsample_bytree=0.5,
                                                         gamma=0, gpu_id=-1,
                                                         importance_type='gain',
                                                         interaction_constraints=None,
                                                         learning_rate=0.03,
                                                         max_delta_step=0,
                                                         max_depth=3,
                                                         min_child_weight=1,
                                                         missing=nan,
                                                         monotone_constraints=None,
                                                         n_estimators=2000,
                                                         n_jobs=-1, nthread=-1,
                                                         num_parallel_tree=1,
                                                         objective='multi:softprob',
                                                         random_state=42,
                                                         reg_alpha=0, reg_lambda=1,
                                                         scale_pos_weight=None,
                                                         subsample=0.5,
                                                         tree_method=None,
                                                         validate_parameters=False,
                                                         verbosity=None),
                                cv=None, method='sigmoid')
```

```
y_pred = sig_clf.predict_proba(X_train_merge)
print('For values of best alpha = 2000 The train log loss is:',log_loss(y_train_merge, y_pred))
```

```
y_pred = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = 2000 The CV log loss is:',log_loss(y_cv_merge, y_pred))
y_pred = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = 2000 The Test log loss is:',log_loss(y_test_merge, y_pred))
plot_confusion_matrix(y_test_merge, sig_clf.predict(X_test_merge))
```

```
For values of best alpha = 2000 The CV log loss is: 0.022790512154724767
For values of best alpha = 2000 The Test log loss is: 0.035168135056459666
Number of misclassified points  0.6439742410303588
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```

```
---------------------------------------------------- Precision matrix ----------------------------------------------------
```

Adding Pixel Intensities from Byte Files

```
import numpy as np
import os
import codecs
import imageio
```

```
import array
from datetime import datetime as dt

if not os.path.isdir("byteFileImages"):
    os.mkdir("byteFileImages")

bytefile_list=os.listdir("byteFiles/")

#This function will generate images from ASM files and Byte files.
def get_images_from_text(filename_list, save_folder):
    for filename in tqdm(filename_list):

        if(filename.endswith("bytes")):
            file = open("byteFiles/"+filename, 'rb')
            file_size = os.path.getsize("byteFiles/"+filename)

        file_width = int(file_size**0.5)
        rem = file_size%file_width
        img_array = array.array('B')
        img_array.fromfile(file,file_size-rem)
        file.close()
        img_arr_final = np.reshape(img_array[:file_width * file_width], (file_width, file_width))
        img_arr_final = np.uint8(img_arr_final)
        imageio.imsave(save_folder+'/' + filename.split(".")[0] + '.png',img_arr_final)
```

Resize the Byte image files to 256x256 dimensions. Keep the original images untouched.

```
from tqdm import tqdm
get_images_from_text(bytefile_list,'byteFileImages')
```

```
100%|██████████| 10868/10868 [1:56:39<00:00,  1.55it/s]
```

```
import cv2
import os
import imageio
```

```
import imageio
from tqdm import tqdm
import random

#Take any image as input, resize it to 256x256, return the resized image
def resize_image(image):
    resized_image = cv2.resize(image, (256,256), interpolation = cv2.INTER_AREA) #Resize all the images to 256X256 dimensions
    return resized_image
```

Display some sample images that we have obtained from Byte files.

```
image_dir = "byteFileImages/"
filenames=random.sample(os.listdir(image_dir),26)

#Display 25 images from BYTE files
plt.figure(figsize=(15,15))
for i in range(1,len(filenames)):
    row = i
    image = imageio.imread(image_dir+filenames[i]) #Image(filename=image_dir+filenames[i])
    image = resize_image(image)
    plt.subplot(5,5,row)
    plt.imshow(image)
plt.show()
```

Get first 800 pixel information from Byte File Images

```
file_lists=os.listdir('byteFileImages/')

with open("features_byte_image_df.csv", mode='w') as byte_image_df: #file_lists = 10868, byte_image_df=800
    byte_image_df.write(','.join(map(str, ["ID"]+["Pixel{}".format(i) for i in range(800)])))
    byte_image_df.write('\n')

    for image in tqdm(file_lists):
        file_id=image.split(".")[0]
        image_array=imageio.imread("byteFileImages/"+image) #This will contain the image matrix in 2D format
        image_array=image_array.flatten()[:800] #Taking the first 1000 pixels from the flattened array
        byte_image_df.write(','.join(map(str, [file_id]+list(image_array))))
        byte_image_df.write('\n') #Write to the dataframe
```

```
100%|██████████| 10868/10868 [08:54<00:00, 20.33it/s]
```

```
byte_image_df=pd.read_csv("features_byte_image_df.csv")
byte_image_df.head()
```

	ID	Pixel0	Pixel1	Pixel2	Pixel3	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8	...	Pixel790	Pixel791	Pix
0	D5ecMC0KmzORPVrwgLT1	48	48	52	48	49	48	48	48	32	...	51	54	
1	JahiBscoyeldwpEV0mxW	48	48	52	48	49	48	48	48	32	...	53	68	
2	0EL7OGZKozbiNCVP61gk	48	48	52	48	49	48	48	48	32	...	56	68	
3	axWR038AmVMhjZOcF6nT	48	48	52	48	49	48	48	48	32	...	70	51	
4	6ZviYOFyQP8sReVI2HrG	48	48	52	48	49	48	48	48	32	...	70	70	

5 rows × 801 columns

```
!rm -r -f byteFiles/ # Deleting bytes files in order to have memory/space for asm files
```

```
!rm -r -f byteFileImages/ # Deleting bytes fileImages in order to have memory/space for asm files
```

Adding Pixel Intensities from ASM Files

```
start=datetime.now()

!7z e train.7z -o/content/asmfiles *.asm -r

end=datetime.now()
difference=end-start
print('Time taken to extract all the asm files',difference)
```

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Xeon(R) CPU @ 2.20GHz (406F0),ASM,AES-NI)

Scanning the drive for archives:
1 file, 18810691091 bytes (18 GiB)

Extracting archive: train.7z
--
Path = train.7z
Type = 7z
Physical Size = 18810691091
Headers Size = 339764
Method = LZMA:24
Solid = +
Blocks = 94

Everything is Ok

Files: 10868
Size:       147076181824
Compressed: 18810691091
Time taken to extract all the asm files 0:28:19.642478
```

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

asmfile_list=os.listdir("asmfiles/")

#This function will generate images from ASM files and Byte files.
def get_images_from_text(filename_list, save_folder):
    for filename in tqdm(filename_list):

        if(filename.endswith("asm")):
            file = codecs.open("asmfiles/"+filename, 'rb')
            file_size = os.path.getsize("asmfiles/"+filename)
            os.remove("asmfiles/"+filename)

        file_width = int(file_size**0.5)
        rem = file_size%file_width
        img_array = array.array('B')
        img_array.fromfile(file,file_size-rem)
        file.close()
        img_arr_final = np.reshape(img_array[:file_width * file_width], (file_width, file_width))
        img_arr_final = np.uint8(img_arr_final)
        imageio.imsave(save_folder+'/' + filename.split(".")[0] + '.png',img_arr_final)


get_images_from_text(asmfile_list,'asmFileImages')
```

```
100%|██████████| 10868/10868 [3:26:35<00:00,  1.14s/it]
```

Display some sample images that we have obtained from ASM files.

```
image_dir = "asmFileImages/"
filenames=random.sample(os.listdir(image_dir),26)

#Display 25 images from ASM files
plt.figure(figsize=(15,15))
```

```
for i in range(1,len(filenames)):
    row = i
    image = imageio.imread(image_dir+filenames[i])
    image = resize_image(image)
    plt.subplot(5,5,row)
    plt.imshow(image)
plt.show()
```

Get first 800 pixel information from ASM File Images

```
file_lists=os.listdir('asmFileImages/')

with open("features_asm_image_df.csv", mode='w') as asm_image_df: #file_lists = 10868, asm_image_df=800
    asm_image_df.write(','.join(map(str, ["ID"]+["Pixel{}".format(i) for i in range(800)])))
    asm_image_df.write('\n')

    for image in tqdm(file_lists):
        file_id=image.split(".")[0]
        image_array=imageio.imread("asmFileImages/"+image) #This will contain the image matrix in 2D format
        image_array=image_array.flatten()[:800] #Taking the first 1000 pixels from the flattened array
        asm_image_df.write(','.join(map(str, [file_id]+list(image_array))))
        asm_image_df.write('\n') #Write to the dataframe
```

```
100%|██████████| 10868/10868 [17:50<00:00, 10.16it/s]
```

```
asm_image_df=pd.read_csv("features_asm_image_df.csv")
```

```
asm_image_df.head()
```

	ID	Pixel0	Pixel1	Pixel2	Pixel3	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8	...	Pixel790	Pixel791	Pi
0	D5ecMC0KmzORPVrwgLT1	46	122	101	110	99	58	48	48	52	...	97	108	
1	JahiBscoyeldwpEV0mxW	72	69	65	68	69	82	58	48	48	...	61	61	
2	0EL7OGZKozbiNCVP61gk	72	69	65	68	69	82	58	48	48	...	61	61	
3	axWR038AmVMhjZOcF6nT	72	69	65	68	69	82	58	48	48	...	61	61	
4	6ZviYOFyQP8sReVI2HrG	72	69	65	68	69	82	58	48	48	...	10	72	

5 rows × 801 columns

Merging of Pixel Intensities from Byte and ASM Files

```
asm_image = pd.read_csv('features_asm_image_df.csv')
asm_image.head()
```

	ID	Pixel0	Pixel1	Pixel2	Pixel3	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8	...	Pixel790	Pixel791	Pi
0	D5ecMC0KmzORPVrwgLT1	46	122	101	110	99	58	48	48	52	...	97	108	
1	JahiBscoyeldwpEV0mxW	72	69	65	68	69	82	58	48	48	...	61	61	
2	0EL7OGZKozbiNCVP61gk	72	69	65	68	69	82	58	48	48	...	61	61	
3	axWR038AmVMhjZOcF6nT	72	69	65	68	69	82	58	48	48	...	61	61	
4	6ZviYOFyQP8sReVI2HrG	72	69	65	68	69	82	58	48	48	...	10	72	

5 rows × 801 columns

```
byte_image = pd.read_csv('features_byte_image_df.csv')
byte_image.head()
```

	ID	Pixel0	Pixel1	Pixel2	Pixel3	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8	...	Pixel790	Pixel791	Pix
0	D5ecMC0KmzORPVrwgLT1	48	48	52	48	49	48	48	48	32	...	51	54	
1	JahiBscoyeldwpEV0mxW	48	48	52	48	49	48	48	48	32	...	53	68	
2	0EL7OGZKozbiNCVP61gk	48	48	52	48	49	48	48	48	32	...	56	68	
3	axWR038AmVMhjZOcF6nT	48	48	52	48	49	48	48	48	32	...	70	51	
4	6ZviYOFyQP8sReVI2HrG	48	48	52	48	49	48	48	48	32	...	70	70	

5 rows × 801 columns

```
final_df_2 = pd.merge(final_df_1,asm_image,on='ID', how='left')
final_df_2 = pd.merge(final_df_2,byte_image,on = 'ID',how = 'left')
print('shape of the dataframe is : ',final_df_2.shape)
```

```
shape of the dataframe is :  (10868, 2912)
```

```
final_df_2.head()
```

```
X = final_df_2.drop(['ID','rtn','.BSS:','.CODE','Class','Unnamed: 0'], axis=1)
y = final_df_1['Class']
print("shape of X : ",X.shape)
print('Shape of y : ',y.shape)
```

```
shape of X :  (10868, 2906)
Shape of y :  (10868,)
```

Train and Test split

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify=y,test_size = 0.20)
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,stratify=y_train,test_size = 0.20)
```

```
print('Shape of X_train : ',X_train.shape[0] )
print('Shape of X_cv : ',X_cv.shape[0] )
print('Shape of X_test : ',X_test.shape[0] )
```

```
Shape of X_train :  6955
Shape of X_cv :  1739
Shape of X_test :  2174
```

Distribution of Each Class

```
train_class_distribution = y_train.value_counts().sort_index()
test_class_distribution = y_test.value_counts().sort_index()
cv_class_distribution = y_cv.value_counts().sort_index()

my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
```

```
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_class_distribution.valu


print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_class_distribution.values

print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
```

```
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_distribution.values[i]/
```

```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
--------------------------------------------------------------------------------
```

Distribution of yi in test data

```
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
--------------------------------------------------------------------------------
```

RandomForest Classifier: Hyperparameter Tuning

```
alpha = [10,50,100,500,1000,2000,3000]
cv_error = []

for i in range(len(alpha)):
    rf_clf = RandomForestClassifier(n_estimators=alpha[i],random_state= 42,n_jobs=-1)
    rf_clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(rf_clf,method ='sigmoid')
    sig_clf.fit(X_train,y_train)
    y_pred = sig_clf.predict_proba(X_cv)
    loss = log_loss(y_cv,y_pred, labels = rf_clf.classes_ ,  eps=1e-15)
    cv_error.append(loss)

for j in range(len(cv_error)):
    print('log_loss for c = ',alpha[j],'is',cv_error[j])
```

```
log_loss for c =  10 is 0.037257985792751996
log_loss for c =  50 is 0.031854693140959414
log_loss for c =  100 is 0.030528093787655898
log_loss for c =  500 is 0.031173265332663555
log_loss for c =  1000 is 0.031228392190550378
log_loss for c =  2000 is 0.03157162511377443
log_loss for c =  3000 is 0.03145013207245903
Number of data points in class 1 : 247 ( 14.264 %)
```

```
best_alpha = np.argmin(cv_error)
plt.plot(alpha,cv_error,label = 'logloss')
plt.legend()
plt.grid()
plt.title('Cross valudation error for each epoch')
plt.xlabel("Alpha Values")
plt.ylabel("Error measure")
plt.show()
```

```
print('Best alph value is : ',alpha[best_alpha])
```

```
Best alph value is :  100
```

```
from sklearn.model_selection import RandomizedSearchCV
model = RandomForestClassifier()
parameters= {
    'n_estimators':[10,50,100,500,1000,2000,3000],
    'max_depth' : [5, 10, 20, 30, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
clf_rs= RandomizedSearchCV(model,param_distributions=parameters,cv=10,scoring= 'neg_log_loss',n_jobs=-1,return_train_score=True)
clf_rs.fit(X_train_merge,y_train_merge)
```

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_job...
                                                    oob_score=False,
                                                    random_state=None,
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [5, 10, 20, 30, 50],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [10, 50, 100, 500, 1000,
                                                         2000, 3000]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='neg_log_loss', verbose=0)
```

```
clf_rs.best_params_
```

```
{'n_estimators': 2000,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_depth': 30}
```

```
clf =  RandomForestClassifier(n_estimators= 2000,min_samples_split= 5,min_samples_leaf= 1,max_depth= 30,n_jobs=-1)
clf.fit(X_train,y_train)
```

```
sig_clf=CalibratedClassifierCV(clf,method='sigmoid')
sig_clf.fit(X_train,y_train)
```

```
CalibratedClassifierCV(base_estimator=RandomForestClassifier(bootstrap=True,
                                                             ccp_alpha=0.0,
                                                             class_weight=None,
                                                             criterion='gini',
                                                             max_depth=30,
                                                             max_features='auto',
                                                             max_leaf_nodes=None,
                                                             max_samples=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=5,
                                                             min_weight_fraction_leaf=0.0,
                                                             n_estimators=2000,
                                                             n_jobs=-1,
                                                             oob_score=False,
                                                             random_state=None,
                                                             verbose=0,
                                                             warm_start=False),
                       cv=None, method='sigmoid')
```

```
y_pred = sig_clf.predict_proba(X_train)
print('For values of best alpha = 2000 The train log loss is:',log_loss(y_train, y_pred))
y_pred = sig_clf.predict_proba(X_cv)
print('For values of best alpha = 2000 The CV log loss is:',log_loss(y_cv, y_pred))
y_pred = sig_clf.predict_proba(X_test)
print('For values of best alpha = 2000 The Test log loss is:',log_loss(y_test, y_pred))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
For values of best alpha = 2000 The CV log loss is: 0.03156590463276551
For values of best alpha = 2000 The Test log loss is: 0.02307083014991414
Number of misclassified points  0.3219871205151794
-------------------------------------------------- Confusion matrix --------------------------------------------------
```

Predicted Class

```
-------------------------------------------------- Precision matrix --------------------------------------------------
```

```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix --------------------------------------------------
```

XGBoost Classifier: Hyperparameter Tuning

```
alpha = [10,50,100,500,1000,2000,3000]
cv_error = []

for i in range(len(alpha)):
    rf_clf = XGBClassifier(n_estimators=alpha[i],random_state= 42,nthread=-1)
    rf_clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(rf_clf,method ='sigmoid')
    sig_clf.fit(X_train,y_train)
    y_pred = sig_clf.predict_proba(X_cv)
    loss = log_loss(y_cv,y_pred, labels = rf_clf.classes_ ,  eps=1e-15)
    cv_error.append(loss)

for j in range(len(cv_error)):
    print('log_loss for c = ',alpha[j],'is',cv_error[j])
```

```
 log_loss for c =  10 is 0.022492927633639193
 log_loss for c =  50 is 0.01899292757168133
 log_loss for c =  100 is 0.01876443468738838
```

```
log_loss for c =  500 is 0.019035230176647752
log_loss for c =  1000 is 0.01903554435664826
log_loss for c =  2000 is 0.019035360426501023
log_loss for c =  3000 is 0.019035299422608877
```

```
best_alpha = np.argmin(cv_error)
plt.plot(alpha,cv_error,label = 'logloss')
plt.legend()
plt.grid()
plt.title('Cross valudation error for each epoch')
plt.xlabel("Alpha Values")
plt.ylabel("Error measure")
plt.show()
print('Best aplha value is: ',alpha[best_alpha])
```