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:_
.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@
.text:0040100D C7 06 08    BB 42 00                       mov     dword ptr [esi],
.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],
.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],
.text:00401039 E8 13 1C    00 00                           call    sub_402C51
.text:0040103E F6 44 24    08 01                           test    byte ptr    [esp+
.text:00401043 74 09                               jz      short loc_40104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E    00 00                           call    ??3@YAXPAX@Z    ;
.text:0040104B 83 C4 04                            add     esp, 4
.text:0040104E
.text:0040104E                            loc_40104E:                     ; CODE XREF:
.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
```

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."

```
# !pip --version
# import matplotlib
# matplotlib.__version__
```

3. Exploratory Data Analysis

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
# matplotlib.use('nbAgg')
%matplotlib notebook
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
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize
from nltk.util import ngrams
import h5py
import copy


#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 w
if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)


# os.makedirs('abc')
# shutil.move('a/abc.txt', 'abc/abc.txt')

# if we have folder called 'train' (train folder contains both .asm files and .bytes files
# for every file that we have in our 'asmFiles' directory we check if it is ending with .b
# '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

```
# close previous plot to render new one
```

```
# close previous plot to render new one
plt.close()
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_heig

#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_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())
```

```
                         ID      size  Class
     0  FGCV8wbAuQp1HiSd7kvm  3.972656      7
     1  EmVYF0ZrIBWCGL5168XH  0.644531      1
     2  CywHbj2cUaVYgp9xhIDk  0.222656      6
     3  eF5UtAWbwBrYJlRVpNIk  0.363281      8
     4  5scq0VdirPRF8DxUKbh1  0.632812      9
```

```
# write size data in csv file
data_size_byte.to_csv('data_size_byte.csv')
```

```
data_size_byte = pd.read_csv('data_size_byte.csv')
print (data_size_byte.head())
```

```
        Unnamed: 0                    ID      size  Class
     0           0  FGCV8wbAuQp1HiSd7kvm  3.972656      7
     1           1  EmVYF0ZrIBWCGL5168XH  0.644531      1
     2           2  CywHbj2cUaVYgp9xhIDk  0.222656      6
     3           3  eF5UtAWbwBrYJlRVpNIk  0.363281      8
     4           4  5scq0VdirPRF8DxUKbh1  0.632812      9
```

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

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

3.2.3 feature extraction from byte files

```
# time taking task, take approx. 30-45 mins
#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,
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)
```

	ID	0	1	2	3	4	5	6	7	8	...	f
0	FGCV8wbAuQp1HiSd7kvm	5784	58	24	32	26	22	7	3	111	...	2
1	EmVYF0ZrIBWCGL5168XH	24597	1258	747	907	1447	566	589	719	1117	...	89

2 rows × 258 columns

```
data_size_byte.head(2)
```

	Unnamed: 0	ID	size	Class
0	0	FGCV8wbAuQp1HiSd7kvm	3.972656	7
1	1	EmVYF0ZrIBWCGL5168XH	0.644531	1

```
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)
```

	ID	0	1	2	3	4	5	6	7	8	...	f
0	FGCV8wbAuQp1HiSd7kvm	5784	58	24	32	26	22	7	3	111	...	
1	EmVYF0ZrIBWCGL5168XH	24597	1258	747	907	1447	566	589	719	1117	...	62

2 rows × 261 columns

```
# 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_valu
    return result1
result = normalize(byte_features_with_size)
```

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

	ID	0	1	2	3	4	5
0	FGCV8wbAuQp1HiSd7kvm	0.002525	0.000082	0.000013	0.000017	0.000016	0.000012
1	EmVYF0ZrIBWCGL5168XH	0.010740	0.001771	0.000416	0.000489	0.000886	0.000320

2 rows × 261 columns

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

	ID	0	1	2	3	4	5
0	FGCV8wbAuQp1HiSd7kvm	0.002525	0.000082	0.000013	0.000017	0.000016	0.000012
1	EmVYF0ZrIBWCGL5168XH	0.010740	0.001771	0.000416	0.000489	0.000886	0.000320
2	CywHbj2cUaVYgp9xhIDk	0.005374	0.000624	0.000130	0.000249	0.000129	0.000100
3	eF5UtAWbwBrYJlRVpNIk	0.008818	0.000957	0.000176	0.000247	0.000174	0.000207
4	5scq0VdirPRF8DxUKbh1	0.037465	0.000991	0.000251	0.000246	0.000315	0.000366

5 rows × 261 columns

3.2.4 Multivariate Analysis

```
plt.close()
#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()
```

```
plt.close()
#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 '
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), d
# split the train data into train and cross validation by maintaining same distribution of
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_siz
```

```
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])
```

```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

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

# my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black, etc.
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], '
```

```
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 %)
```

```
plt.close()
# my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black, etc.
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()
```

```
plt.show()

print('-'*80)
# 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], '(
```

```
--------------------------------------------------------------------------------
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 %)
```

```
plt.close()
# my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black, etc.
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()
```

```
print('-'*80)
# 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], '(',
```

```
--------------------------------------------------------------------------------
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 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

```
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

    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
# 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
# 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=label
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=label
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=label
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,


# 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-

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

```
Log loss on Cross Validation Data using Random Model 2.4765227422803386
Log loss on Test Data using Random Model 2.5086309644540004
Number of misclassified points  88.9604415823367
---------------------------------------------------- Confusion matrix -----------------
```

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

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

4.1.2. K Nearest Neighbour Classification

```
%%time
plt.close()
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/gener
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p
# 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-ne
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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()
```

```
log_loss for k =  1 is 0.22197129792560755
log_loss for k =  3 is 0.2089185330714807
log_loss for k =  5 is 0.22690358877552813
log_loss for k =  7 is 0.24413578025615393
log_loss for k =  9 is 0.25968782214603464
log_loss for k =  11 is 0.2764987380541104
log_loss for k =  13 is 0.2904920581051572
```

```
CPU times: user 57.8 s, sys: 0 ns, total: 57.8 s
Wall time: 57.8 s
```

```
# %%time
plt.close()
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)
```

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

```
For values of best alpha =  3 The train log loss is: 0.13309130472155042
For values of best alpha =  3 The cross validation log loss is: 0.2089185330714807
For values of best alpha =  3 The test log loss is: 0.20041775276335702
Number of misclassified points  5.105795768169273
--------------------------------------------------- Confusion matrix -----------------
```

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

4.1.3. Logistic Regression

```
%%time
plt.close()
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# predict(X)  Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geom
#------------------------------

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-

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()
```

```
log_loss for c =  1e-05 is 1.5750350606589754
log_loss for c =  0.0001 is 1.5790516337094596
log_loss for c =  0.001 is 1.5431918470950965
log_loss for c =  0.01 is 1.0242786142597624
log_loss for c =  0.1 is 0.8602633724481396
log_loss for c =  1 is 0.7232469080407065
log_loss for c =  10 is 0.5905983721266695
log_loss for c =  100 is 0.5378658285030614
log_loss for c =  1000 is 0.6184257388607202
```

```
%%time
plt.close()
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_, e
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-
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_, eps
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log loss for train data 0.5161568454325155
log loss for cv data 0.5378658285030614
log loss for test data 0.5287814318994476
Number of misclassified points  12.051517939282428
---------------------------------------------------- Confusion matrix -----------------
```

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

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

4.1.4. Random Forest Classifier

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------

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()
```

```
log_loss for c =  10 is 0.10832075575112872
log_loss for c =  50 is 0.08807820249822448
log_loss for c =  100 is 0.08788476126650518
log_loss for c =  500 is 0.08584374512867923
log_loss for c =  1000 is 0.08549101437528983
log_loss for c =  2000 is 0.08503123176112597
log_loss for c =  3000 is 0.08498094708786469
```

```
CPU times: user 29min 31s, sys: 18.4 s, total: 29min 49s
Wall time: 4min 17s
```

```
plt.close()
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(
```

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

```
For values of best alpha =  3000 The train log loss is: 0.028706691980921458
For values of best alpha =  3000 The cross validation log loss is: 0.084980947087864
For values of best alpha =  3000 The test log loss is: 0.06130664008293945
Number of misclassified points  1.3339466421343145
--------------------------------------------------- Confusion matrix -----------------
```

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

4.1.5. XgBoost Classification

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
```

```
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/reg
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

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()
```

```
log_loss for c =  10 is 0.19005861435310809
log_loss for c =  50 is 0.09300883717009732
log_loss for c =  100 is 0.0706585262688187
log_loss for c =  500 is 0.06197621824496104
log_loss for c =  1000 is 0.0613439706923832
log_loss for c =  2000 is 0.06159227398299267
```

```
%%time
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
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  1000 The train log loss is: 0.023647153426002675
For values of best alpha =  1000 The cross validation log loss is: 0.061343970692383
For values of best alpha =  1000 The test log loss is: 0.07842603813981379
CPU times: user 31min 41s, sys: 56.6 s, total: 32min 37s
Wall time: 4min 5s
```

```
plt.close()
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
Number of misclassified points  1.3799448022079117
-------------------------------------------------- Confusion matrix ----------------
```

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

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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-wit
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)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   26.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  3.6min remaining:  2.1min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  6.4min remaining:  1.9min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 10.5min remaining:  1.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 11.5min finished
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
```

```
                                                seed=None, silent=None, subsample=1,
                                                verbosity=1),
                          iid='warn', 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=False, scoring=None, verbose=10)
```

```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.15, 'colsam
```

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])  Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.15, colsample_bytree=0.3, max_depth
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))
```

```
train loss 0.023248932222341635
cv loss 0.0697311485102128
test loss 0.08183939406086423
```

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 us


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
#thrid
#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]],'thrid')
        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:',
        #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', 'd
        #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:',
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'd
    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:',
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'd
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')
    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('thrid/'+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:',
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'd
    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:',
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'd
    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 extrac
# 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()
```

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0

5 rows × 53 columns

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_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())
```

```
                         ID      size  Class
     0  4sChO0eqidfLtS83JMVE  0.233131      3
     1  89VdQqEkGB2Fzv4lXnbU  0.121401      3
     2  3hfRptP5gUEqDHlCkMYX  0.360250      4
     3  6GgdCXPL2Bali0f8ZFkA  0.122087      3
     4  E4SvkGO5CuB8LehadFo6  0.122047      3
```

4.2.1.2 Distribution of .asm file sizes

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

```
# 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()
```

```
(10868, 53)
(10868, 3)
```

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0

5 rows × 54 columns

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

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rda
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

```
plt.close()
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
```

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

```
plt.close()
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
```

```
plt.close()
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
```

```
plt.close()
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
```

```
plt.close()
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
```

```
plt.close()
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
```

```
plt.close()
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.
```

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

boxplot of .asm push opcode

```
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

```
plt.close()
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-st

#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()
```

```
plt.close()
# by univariate analysis on the .asm file features we are getting very negligible informat
# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after remo
# the plot looks very messy

xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.CODE','size']
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=
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,s
```

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

```
HEADER:    False
.text:     False
.Pav:      False
.idata:    False
.data:     False
.bss:      False
.rdata:    False
.edata:    False
.rsrc:     False
.tls:      False
.reloc:    False
jmp        False
mov        False
retf       False
push       False
pop        False
xor        False
retn       False
nop        False
sub        False
inc        False
dec        False
add        False
imul       False
xchg       False
or         False
shr        False
cmp        False
call       False
shl        False
ror        False
rol        False
jnb        False
jz         False
lea        False
movzx      False
.dll       False
std::      False
:dword     False
edx        False
```

```
esi        False
eax        False
ebx        False
ecx        False
edi        False
ebp        False
esp        False
eip        False
size       False
dtype: bool
```

4.4. Machine Learning models on features of .asm files

4.4.1 K-Nearest Neigbors

```
%%time
plt.close()
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/gener
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p
# 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-ne
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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-

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()
```

```
log_loss for k =  1 is 0.09551157959211777
log_loss for k =  3 is 0.09805566398432622
```

```
%%time
plt.close()
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))
```

```
log loss for train data 0.037475912101338785
log loss for cv data 0.09551157959211777
log loss for test data 0.07985178727072437
Number of misclassified points  1.1039558417663293
---------------------------------------------------- Confusion matrix -----------------
```

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

4.4.2 Logistic Regression

```
%%time
plt.close()
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geom
#------------------------------


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

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()
```

```
log_loss for c =  1e-05 is 1.5661498686463826
log_loss for c =  0.0001 is 1.517675047020921
log_loss for c =  0.001 is 1.2191622564580826
log_loss for c =  0.01 is 1.1642755471095085
log_loss for c =  0.1 is 1.0393439618593712
log_loss for c =  1 is 0.7399622931531011
log_loss for c =  10 is 0.5521355759037754
log_loss for c =  100 is 0.4561736339097347
log_loss for c =  1000 is 0.40949221859900775
```

```
%%time
plt.close()
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.classe
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_, ep
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_
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log loss for train data 0.3951231421379966
log loss for cv data 0.40949221859900775
log loss for test data 0.39929985766626985
Number of misclassified points  8.693652253909843
---------------------------------------------------- Confusion matrix -----------------
```

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

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

4.4.3 Random Forest Classifier

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------

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)
    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-

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()
```

```
log_loss for c =  10 is 0.04356250797901248
log_loss for c =  50 is 0.037363030786582735
log_loss for c =  100 is 0.036614779171787376
```

```
%%time
plt.close()
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_
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=
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_,
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log loss for train data 0.021489116411912302
log loss for cv data 0.036614779171787376
log loss for test data 0.03239270498531275
Number of misclassified points  0.5059797608095675
---------------------------------------------------- Confusion matrix -----------------
```

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

4.4.4 XgBoost Classifier

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

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-

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()
```

```
log_loss for c =  10 is 0.11212661518377143
log_loss for c =  50 is 0.05582972313723833
log_loss for c =  100 is 0.03525743616432047
log_loss for c =  500 is 0.031322696772212756
log_loss for c =  1000 is 0.031566884648293325
log_loss for c =  2000 is 0.03153641864350838
log_loss for c =  3000 is 0.03140732250886729
```

```
%%time
# plt.close()
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
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  500 The train log loss is: 0.02020393568415983
For values of best alpha =  500 The cross validation log loss is: 0.0313226967722127
For values of best alpha =  500 The test log loss is: 0.030377059563789385
CPU times: user 1min 48s, sys: 0 ns, total: 1min 48s
Wall time: 1min 48s
```

```
plt.close()
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
Number of misclassified points  0.45998160073597055
-------------------------------------------------- Confusion matrix -----------------
```

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

4.4.5 Xgboost Classifier with best hyperparameters

```
%%time
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)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   12.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  1.6min remaining:   56.6s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.9min remaining:   34.1s
```

```
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  2.0min remaining:   13.4s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.2min finished
CPU times: user 14.5 s, sys: 323 ms, total: 14.8 s
Wall time: 2min 24s
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', 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=False, scoring=None, verbose=10)
```

```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.15, 'colsamp
```

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])  Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

x_cfl=XGBClassifier(n_estimators=500,subsample=1,learning_rate=0.15,colsample_bytree=0.3,m
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))
```

```
train loss 0.019288923540468395
cv loss 0.03159265585460439
test loss 0.02576461657919649
```

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()
```

	ID	0	1	2	3	4	5
0	FGCV8wbAuQp1HiSd7kvm	0.002525	0.000082	0.000013	0.000017	0.000016	0.000012
1	EmVYF0ZrIBWCGL5168XH	0.010740	0.001771	0.000416	0.000489	0.000886	0.000320
2	CywHbj2cUaVYgp9xhIDk	0.005374	0.000624	0.000130	0.000249	0.000129	0.000100
3	eF5UtAWbwBrYJlRVpNIk	0.008818	0.000957	0.000176	0.000247	0.000174	0.000207
4	5scq0VdirPRF8DxUKbh1	0.037465	0.000991	0.000251	0.000246	0.000315	0.000366

5 rows × 261 columns

```
result_asm.head()
```

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rda
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000

5 rows × 54 columns

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

```
(10868, 261)
(10868, 54)
```

```
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()
```

	0	1	2	3	4	5	6	7	
0	0.002525	0.000082	0.000013	0.000017	0.000016	0.000012	0.000004	0.000003	0.000
1	0.010740	0.001771	0.000416	0.000489	0.000886	0.000320	0.000332	0.000662	0.000
2	0.005374	0.000624	0.000130	0.000249	0.000129	0.000100	0.000105	0.000168	0.000
3	0.008818	0.000957	0.000176	0.000247	0.000174	0.000207	0.000123	0.000221	0.000
4	0.037465	0.000991	0.000251	0.000246	0.000315	0.000366	0.000252	0.000369	0.000

5 rows × 308 columns

4.5.2. Multivariate Analysis on final fearures

```
plt.close()

xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform((result_x))
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,stratif
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,s
```

4.5.4. Random Forest Classifier on final features

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------

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=1

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()
```

```
log_loss for c =  10 is 0.056458997763404746
log_loss for c =  50 is 0.04409187353003727
log_loss for c =  100 is 0.044061068765813394
log_loss for c =  500 is 0.042874337981406434
log_loss for c =  1000 is 0.042932057098188925
log_loss for c =  2000 is 0.04296267385152051
log_loss for c =  3000 is 0.043147743596854915
```

```
CPU times: user 26min 28s, sys: 18.4 s, total: 26min 46s
Wall time: 3min 59s
```

```
%%time
plt.close()
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  500 The train log loss is: 0.014762155486539353
For values of best alpha =  500 The cross validation log loss is: 0.0428743379814064
For values of best alpha =  500 The test log loss is: 0.04192004241532452
CPU times: user 2min 3s, sys: 1.97 s, total: 2min 5s
Wall time: 20.2 s
```

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

```
Number of misclassified points  0.9199632014719411
-------------------------------------------------- Confusion matrix -----------------
```

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

4.5.5. XgBoost Classifier on final features

```
-------------------------------------------------- Recall matrix --------------------
```

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

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=1

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()
```

```
log_loss for c =  10 is 0.1071910897830746
log_loss for c =  50 is 0.06031120083021808
log_loss for c =  100 is 0.052799286967522965
log_loss for c =  500 is 0.0511063291087567
log_loss for c =  1000 is 0.05132551659041581
log_loss for c =  2000 is 0.050734482379753906
log_loss for c =  3000 is 0.05075047417606582
```

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  2000 The train log loss is: 0.009815691068426237
For values of best alpha =  2000 The cross validation log loss is: 0.050750474176065
For values of best alpha =  2000 The test log loss is: 0.042754901893074994
CPU times: user 42min 9s, sys: 56 ms, total: 42min 9s
Wall time: 42min 9s
```

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

```
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)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   46.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  7.8min remaining:  4.5min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  8.0min remaining:  2.4min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  9.5min remaining:  1.1min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 10.3min finished
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', 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=False, scoring=None, verbose=10)
```

```
print (random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.05, 'cols
```

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
```

```
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

x_cfl=XGBClassifier(n_estimators=2000,max_depth=3,learning_rate=0.05,colsample_bytree=0.5,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  2000 The train log loss is: 0.010227838888593169
For values of best alpha =  2000 The cross validation log loss is: 0.046105797578046
For values of best alpha =  2000 The test log loss is: 0.03857589259645944
CPU times: user 21min 21s, sys: 120 ms, total: 21min 21s
Wall time: 21min 21s
```

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

```
Number of misclassified points  0.78196872125115
-------------------------------------------------- Confusion matrix -----------------
```

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

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

Bytes Files Model Comparision

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = (['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', 'Number o

table.add_row(['Random Model', '-', 2.4765227422803386, 2.5086309644540004, 88.96044158233
table.add_row(['K Nearest Neighbour', 0.13309130472155042, 0.2089185330714807, 0.200417752
table.add_row(['Logistic Regression', 0.5161568454325155, 0.5378658285030614, 0.5287814318
table.add_row(['Random Forest Classifier', 0.028706691980921458, 0.08498094708786469, 0.06
table.add_row(['XgBoost Classification', 0.023248932222341635, 0.0697311485102128, 0.08183
print(table)
```

```
+-------------------------+----------------------+----------------------+------------
|          Model          |    Train Log Loss    |     CV Log Loss      |    Test Log
+-------------------------+----------------------+----------------------+------------
|       Random Model      |          -           |  2.4765227422803386  |  2.5086309
|   K Nearest Neighbour   | 0.13309130472155042  |  0.2089185330714807  | 0.200417752
|   Logistic Regression   |  0.5161568454325155  |  0.5378658285030614  |  0.5287814
| Random Forest Classifier | 0.028706691980921458 | 0.08498094708786469 | 0.06130664
|  XgBoost Classification | 0.023248932222341635 |  0.0697311485102128  | 0.08183939
+-------------------------+----------------------+----------------------+------------
```

ASM Files Model Comparision

```
table = PrettyTable()
table.field_names  = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', 'Number o

table.add_row(['K Nearest Neighbour', 0.037475912101338785, 0.09551157959211777, 0.0798517
table.add_row(['Logistic Regression', 0.3951231421379966, 0.40949221859900775, 0.399299857
table.add_row(['Random Forest Classifier', 0.021489116411912302, 0.036614779171787376, 0.0
table.add_row(['XgBoost Classification', 0.019288923540468395, 0.03159265585460439, 0.0257
print(table)
```

```
+-------------------------+----------------------+----------------------+----------
|          Model          |    Train Log Loss    |     CV Log Loss      |    Test Lo
```

```
+-------------------------+----------------------+----------------------+----------
|   K Nearest Neighbour   | 0.037475912101338785 | 0.09551157959211777  | 0.07985178
|   Logistic Regression   |  0.3951231421379966  | 0.40949221859900775  | 0.39929985
| Random Forest Classifier | 0.021489116411912302 | 0.036614779171787376 | 0.03239276
|  XgBoost Classification | 0.019288923540468395 | 0.03159265585460439  | 0.02576461
+-------------------------+----------------------+----------------------+----------
```

Bytes + ASM Files Model Comparision

```
table = PrettyTable()
table.field_names  = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss', 'Number o

table.add_row(['Random Forest Classifier', 0.014762155486539353, 0.042874337981406434, 0.0
table.add_row(['XgBoost Classification', 0.010227838888593169, 0.046105797578046656, 0.038
print(table)
```

```
+-------------------------+----------------------+----------------------+----------
|          Model          |    Train Log Loss    |     CV Log Loss      |    Test Lo
+-------------------------+----------------------+----------------------+----------
| Random Forest Classifier | 0.014762155486539353 | 0.042874337981406434 | 0.04192004
|  XgBoost Classification | 0.010227838888593169 | 0.046105797578046656 | 0.03857589
+-------------------------+----------------------+----------------------+----------
```

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

2. if you are having computation power limitations, you can try using google colab, with

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

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

Assignment Part 1

- 1 Add bi-grams and n-grams on byte files and improve the log-loss

1.1 Process Bigram on ByteFiles

```
# list byteFiles directory
files = os.listdir('byteFiles')
```

```
len(files)
```

```
10868
```

```
# all_keys = []

def calc_bi_grams(file):
    '''
    Calculate unigram and bigram
    '''
    temp_list = []
    with open('byteFiles/'+file,"r") as byte_flie:
        all_lines = []
        for lines in byte_flie:
            line=lines.rstrip().split(" ")
            all_lines.extend(line)
            # unigrams
            for hex_code in line:
                temp_list.append(hex_code.lower())
        temp_list = list(set(temp_list))
        # bigrams
        bi_g = [' '.join(x) for x in list(ngrams(all_lines, 2))]
        for hex_code in bi_g:
            temp_list.append(hex_code.lower())
        temp_list = list(set(temp_list))
    return temp_list
```

```
# Process and collect bigram of bytes file
from tqdm import tqdm
for file in tqdm(files):
    all_keys.extend(calc_bi_grams(file))
    all_keys = list(set(all_keys))

len(all_keys)
```

```
all_keys
```

```
with open('uni_bigram_keys.pkl', 'wb') as big:
    pickle.dump(all_keys, big)
```

```
with open('uni_bigram_keys.pkl', 'rb') as big:
```

```
    all_keys = pickle.load(big)
```

```
len(all_keys)
```

```
66306
```

```
a="00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15\
,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29\
,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,\
3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,\
53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,\
69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,\
80,81,82,83,84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,\
97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,\
ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,\
c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,\
df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,\
f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
a = a.replace(",", " ")
a_uni = a.split(" ")
len(a_uni)
```

```
257
```

```
# all_keys.extend(a_uni)
```

```
len(all_keys)
```

```
66306
```

```
all_keys
```

```
 '05 09',
 '89 4f',
 'a4 d4',
 'd5 5f',
 '08 cd',
 'e8 84',
 '32 5c',
 '1c 1d',
 'cc 23',
 'f9 d6',
 '1e 7c',
 'd7 21',
 '6d 1a',
 '3d e9',
 '69 d9',
 'e5 a3',
 '3f 70',
 '60 bc',
 '74 c3',
 'a0 cc',
 'fd 7b',
 '09 50',
 'bc cd',
```

```
        'f8 02',
        'ad 23',
        '5f 23',
        'e2 d4',
        'e0 6e',
        '59 e6',
        'd3 8b',
        'b1 fe',
        '9e 67',
        '68 f6',
        '30 55',
        'd2 a3',
        '43 7c',
        'c5 d2',
        'c9 5e',
        'ff ff',
        '1d fb',
        'fb d3',
        '9f 15',
        'be c0',
        'fc 86',
        '0a 12',
        'ad 50',
        '48 2b',
        '3a e7',
        'd9 8e',
        '94 e3',
        'd1 3a',
        '86 c2',
        'c0 68',
        'ca 74',
        '8d 9b',
        '86 2a',
        'c8 df',
        '8b 84',
        'ea 07',
```

```
import scipy
from scipy.sparse import csr_matrix
from tqdm import tqdm

# bytebigram_vect = csr_matrix((len(files), len(all_keys)))
# from tqdm import tqdm
# for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
#     f = open('byteFiles/' + file)
#     bytebigram_vect[i, :] += csr_matrix(vector.fit_transform([f.read().replace('\n', ' '
#     f.close()

def extract_bigrams_byte_features(files):
    '''
    Extract Bigram features and save the output file in npz file, This method is called in
    '''
    pid = os.getpid()
    print('Process started with id = ', pid)
    ftot = len(files)

    str_pid = str(pid)
```

```
    vector_str_pid = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=all_key
    bytebigram_vect_str_pid = csr_matrix((len(files), len(all_keys)))
    print('vector variable created with name ', 'bytebigram_vect', '_', str_pid)
    for i, file in enumerate(files):
        f = open('byteFiles/' + file)
        bytebigram_vect_str_pid[i, :] += csr_matrix(vector_str_pid.fit_transform([f.read()
        f.close()
        print(pid, i + 1, 'of', ftot, 'files processed.')

        # Print progress
#         if (i+1) % 100 == 0:
#             print(pid, i + 1, 'of', ftot, 'files processed.')

    filename = 'bytebigram_'+str_pid+'.npz'
    scipy.sparse.save_npz(filename, bytebigram_vect_str_pid)#'bytebigram.npz'

# extract_bigrams_byte_features('abc')


#·Now·divide·the·train·files·into·four·groups·for·multiprocessing,·Six·threads·not·works·p
import·time·as·tm
start_time·=·tm.time()
tfiles·=·os.listdir('byteFiles')
#·1810*6·=·10860·for·last·slot·add·8
#·sixth·=·1810#int(len(tfiles)/4)
#·print(quart)
#·train1·=·tfiles[:sixth]
#·train2·=·tfiles[sixth:(2*sixth)]
#·train3·=·tfiles[(2*sixth):(3*sixth)]
#·train4·=·tfiles[(3*sixth):(4*sixth)]
#·train5·=·tfiles[(4*sixth):(5*sixth)]
#·train6·=·tfiles[(5*sixth):]
#·#·print(len(train6))
#·print(len(tfiles),·sixth,·(len(train1)+len(train2)+len(train3)+len(train4)+len(train5)+l
#·trains·=·[train1,·train2,·train3,·train4,·train5,·train6]
#·p·=·Pool(6)
quart·=·int(len(tfiles)/4)
#·print(quart)
train1·=·tfiles[:quart]
train2·=·tfiles[quart:(2*quart)]
train3·=·tfiles[(2*quart):(3*quart)]
train4·=·tfiles[(3*quart):]
print(len(tfiles),·quart,·(len(train1)+len(train2)+len(train3)+len(train4)))
trains·=·[train1,·train2,·train3,·train4]
p·=·Pool(4)
p.map(extract_bigrams_byte_features,·trains)
print("Elapsed·time:·{:.2f}·hours.".format((tm.time()·-·start_time)/3600.0))
```

```
10868 2717 10868
Process started with id =  2034
Process started with id =  2035
Process started with id =  2036
Process started with id =  2037
vector variable created with name  bytebigram_vect _ 2035
vector variable created with name  bytebigram_vect _ 2034
vector variable created with name  bytebigram_vect _ 2036
```

```
vector variable created with name  bytebigram_vect _ 2037
2034 1 of 2717 files processed.
2034 2 of 2717 files processed.
2034 3 of 2717 files processed.
2034 4 of 2717 files processed.
2036 1 of 2717 files processed.
2035 1 of 2717 files processed.
2034 5 of 2717 files processed.
2036 2 of 2717 files processed.
2034 6 of 2717 files processed.
2037 1 of 2717 files processed.
2036 3 of 2717 files processed.
2035 2 of 2717 files processed.
2036 4 of 2717 files processed.
2035 3 of 2717 files processed.
2037 2 of 2717 files processed.
2034 7 of 2717 files processed.
2034 8 of 2717 files processed.
2035 4 of 2717 files processed.
2037 3 of 2717 files processed.
2037 4 of 2717 files processed.
2036 5 of 2717 files processed.
2037 5 of 2717 files processed.
2036 6 of 2717 files processed.
2035 5 of 2717 files processed.
2036 7 of 2717 files processed.
2036 8 of 2717 files processed.
2034 9 of 2717 files processed.
2035 6 of 2717 files processed.
2034 10 of 2717 files processed.
2034 11 of 2717 files processed.
2036 9 of 2717 files processed.
2034 12 of 2717 files processed.
2037 6 of 2717 files processed.
2034 13 of 2717 files processed.
2035 7 of 2717 files processed.
2037 7 of 2717 files processed.
2035 8 of 2717 files processed.
2034 14 of 2717 files processed.
2036 10 of 2717 files processed.
2037 8 of 2717 files processed.
2035 9 of 2717 files processed.
2036 11 of 2717 files processed.
2035 10 of 2717 files processed.
2034 15 of 2717 files processed.
2035 11 of 2717 files processed.
2036 12 of 2717 files processed.
2036 13 of 2717 files processed.
2037 9 of 2717 files processed.
2037 10 of 2717 files processed.
```

```
# Load all genereated npz files
import scipy
from scipy.sparse import csr_matrix
from sklearn.preprocessing import normalize
bytebigram_vect_1 = scipy.sparse.load_npz('bytebigram_2034.npz')#normalize(scipy.sparse.lo
bytebigram_vect_2 = scipy.sparse.load_npz('bytebigram_2035.npz')#normalize(scipy.sparse.lo
bytebigram_vect_3 = scipy.sparse.load_npz('bytebigram_2036.npz')#normalize(scipy.sparse.lo
bytebigram_vect_4 = scipy.sparse.load_npz('bytebigram_2037.npz')#normalize(scipy.sparse.lo
```

```
# bytebigram_vect_5 = normalize(scipy.sparse.load_npz('bytebigram_8767.npz'), axis = 0)
# bytebigram_vect_6 = normalize(scipy.sparse.load_npz('bytebigram_8768.npz'), axis = 0)


# Merge all npz files
from scipy.sparse import vstack
bytebigram_vect_7 = vstack((bytebigram_vect_1, bytebigram_vect_2, bytebigram_vect_3, byteb
# , bytebigram_vect_5, bytebigram_vect_6

bytebigram_vect_7
```

```
<10868x66306 sparse matrix of type '<class 'numpy.float64'>'
        with 502109160 stored elements in Compressed Sparse Row format>
```

```
def imp_features(data, features, keep):
    '''
    Collect important features using Random Forest Classifier
    '''
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    sns.set()
    plt.figure(figsize = (10, 5))
    ax = sns.barplot(x = imp_feature_name, y = imp_value)
    ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
    sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
    plt.title('Important Features')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')
    return imp_feature_indx[:keep]


#·collect·top·300·features·from·bigrams
byte_bi_indxes_1·=·imp_features(normalize(bytebigram_vect_7,·axis=0),·all_keys,·300)
```

```
data_y.head()
result_y = data_y
```

```
byte_bi_indxes_1
```

```
array([14157, 40691, 17336, 46660,  8825,  4268, 28027, 12130, 21680,
        8953, 23402, 60240, 40185, 57251, 60461, 66125, 28941, 52525,
       11420, 42315,  1194, 52530, 33727, 23362, 38653, 32566,  5260,
       64991, 58535, 35728, 40950, 14695, 28918, 48334, 51662, 15622,
       20587, 13129, 23666, 10273, 12978, 43239,  6174, 51699, 20328,
       10730, 23968, 24865, 29100,  4670, 46546, 57986, 55909, 50088,
       45420, 20401, 14374,  9000,  6345, 38518,  3172, 28171, 35826,
       53280, 61323, 16859, 21166, 26032, 37345, 39969,  3842, 11095,
       16392, 10395, 63477, 63883, 28780, 21058, 12977, 29208,  9070,
       49561, 29935, 52039, 63461, 59348, 16332, 31246, 22790, 37400,
       15607, 25545, 41632, 54259, 63116, 21975, 13175, 43830,  4280,
       50343, 63837, 17033, 50978, 40625, 51685, 30258, 49647, 17990,
       25182, 19723, 42439, 17054, 21857, 51058, 37362, 11483, 54966,
       40121, 51962, 12827, 14293, 21291,  1697, 49129, 54347, 46965,
       55250, 30900, 63464, 60249, 57395, 64040, 38751, 57411, 55023,
       48655, 38375,  8497, 60805, 41133, 48688, 11532, 15447, 14281,
       59816, 35801,  5376, 31260, 26502, 23967,   195, 65467, 20405,
       54663,  5833,  3539, 55511, 16739, 28444, 50815, 24802, 29339,
        6658, 31524, 40996, 56374, 43207, 66184, 31835, 26082, 41514,
       53820, 13307, 17392,  4019, 10927, 23215, 18124, 16169, 32522,
       28977, 31306, 23868, 60693,  2750, 16673, 28529, 31491, 51248,
       26920, 34232, 10409, 47688, 39505, 10089, 55965, 34194, 28953,
        9903, 62172, 53656, 57598, 28603, 18756, 18876, 10725,   185,
       55636,  9957, 36060, 17849, 65135,  9891, 59251, 64398, 39338,
       31351, 33420, 37802, 23257, 37528, 39836, 15681, 33561, 65231,
       18636, 51508, 46730, 39445, 12347, 21445, 26236, 32311,  9668,
       22755, 45469, 52047, 32420,  2785, 29259,  1472, 11015, 34507,
       45665, 57493, 64509,  3154, 42228, 39442, 57594, 14572, 34421,
       54613,  3936, 36977, 14803,  3871, 23382, 40434, 35758, 62469,
       50927, 18393, 24588, 60639, 28632, 20119, 49847, 11640,  2687,
       54923, 25605, 36430, 65573, 13473,  3597, 36518, 62401, 18907,
       46110, 54085, 21236,  5877, 43601, 18066, 14375, 39937, 14102,
       47992, 40069,  6096,  8713, 36692,  1492, 22573, 18440, 40746,
       43090, 23149, 24251])
```

```
# Save generated index file
np.save('byte_bi_indx', byte_bi_indxes_1)
```

```
byte_bi_indxes = np.load('byte_bi_indx.npy')
```

```
byte_bi_indxes
```

```
array([14157, 40691, 17336, 46660,  8825,  4268, 28027, 12130, 21680,
        8953, 23402, 60240, 40185, 57251, 60461, 66125, 28941, 52525,
       11420, 42315,  1194, 52530, 33727, 23362, 38653, 32566,  5260,
       64991, 58535, 35728, 40950, 14695, 28918, 48334, 51662, 15622,
       20587, 13129, 23666, 10273, 12978, 43239,  6174, 51699, 20328,
       10730, 23968, 24865, 29100,  4670, 46546, 57986, 55909, 50088,
       45420, 20401, 14374,  9000,  6345, 38518,  3172, 28171, 35826,
       53280, 61323, 16859, 21166, 26032, 37345, 39969,  3842, 11095,
       16392, 10395, 63477, 63883, 28780, 21058, 12977, 29208,  9070,
       49561, 29935, 52039, 63461, 59348, 16332, 31246, 22790, 37400,
       15607, 25545, 41632, 54259, 63116, 21975, 13175, 43830,  4280,
       50343, 63837, 17033, 50978, 40625, 51685, 30258, 49647, 17990,
       25182, 19723, 42439, 17054, 21857, 51058, 37362, 11483, 54966,
       40121, 51962, 12827, 14293, 21291,  1697, 49129, 54347, 46965,
       55250, 30900, 63464, 60249, 57395, 64040, 38751, 57411, 55023,
       48655, 38375,  8497, 60805, 41133, 48688, 11532, 15447, 14281,
       59816, 35801,  5376, 31260, 26502, 23967,   195, 65467, 20405,
       54663,  5833,  3539, 55511, 16739, 28444, 50815, 24802, 29339,
        6658, 31524, 40996, 56374, 43207, 66184, 31835, 26082, 41514,
       53820, 13307, 17392,  4019, 10927, 23215, 18124, 16169, 32522,
       28977, 31306, 23868, 60693,  2750, 16673, 28529, 31491, 51248,
       26920, 34232, 10409, 47688, 39505, 10089, 55965, 34194, 28953,
        9903, 62172, 53656, 57598, 28603, 18756, 18876, 10725,   185,
       55636,  9957, 36060, 17849, 65135,  9891, 59251, 64398, 39338,
       31351, 33420, 37802, 23257, 37528, 39836, 15681, 33561, 65231,
       18636, 51508, 46730, 39445, 12347, 21445, 26236, 32311,  9668,
       22755, 45469, 52047, 32420,  2785, 29259,  1472, 11015, 34507,
       45665, 57493, 64509,  3154, 42228, 39442, 57594, 14572, 34421,
       54613,  3936, 36977, 14803,  3871, 23382, 40434, 35758, 62469,
       50927, 18393, 24588, 60639, 28632, 20119, 49847, 11640,  2687,
       54923, 25605, 36430, 65573, 13473,  3597, 36518, 62401, 18907,
       46110, 54085, 21236,  5877, 43601, 18066, 14375, 39937, 14102,
       47992, 40069,  6096,  8713, 36692,  1492, 22573, 18440, 40746,
       43090, 23149, 24251])
```

```
# Pick top 300 features
top_byte_bi = np.zeros((10868, 0))
for i in byte_bi_indxes:
    sliced = bytebigram_vect_7[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])


# Create a DataFrame using top 300 features
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns = np.take(all_keys, byte_bi_indxes))


# Save DataFrame into CSV
byte_bi_df.to_dense().to_csv('byte_bi.csv')


# Read csv and start processing the data
byte_bi_df = pd.read_csv('byte_bi.csv').drop('Unnamed: 0', axis = 1)#fillna(0)


byte_bi_df
```

	1d 6b	2e dc	cb 26	d2 86	f3 67	66 8c	8a 46	4e 47	e2 1f	49 4e	...	c3 8b	c4 0c	3b 38	3a 5c
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	3.0	0.0	0.0
1	3.0	3.0	5.0	1.0	1.0	8.0	2.0	287.0	1.0	282.0	...	5.0	11.0	1.0	1.0
2	0.0	0.0	2.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	...	1.0	2.0	2.0	0.0
3	0.0	0.0	0.0	0.0	0.0	4.0	2.0	2.0	0.0	2.0	...	16.0	8.0	2.0	3.0
4	0.0	0.0	0.0	1.0	2.0	0.0	0.0	2.0	1.0	5.0	...	6.0	3.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10863	1.0	1.0	1.0	2.0	0.0	2.0	0.0	3.0	0.0	0.0	...	3.0	5.0	1.0	1.0
10864	2.0	1.0	0.0	1.0	0.0	18.0	41.0	56.0	1.0	56.0	...	121.0	125.0	0.0	12.0
10865	2.0	2.0	1.0	1.0	3.0	14.0	0.0	29.0	0.0	42.0	...	75.0	114.0	6.0	4.0
10866	0.0	0.0	0.0	0.0	0.0	18.0	49.0	1.0	0.0	2.0	...	131.0	48.0	0.0	0.0
10867	0.0	0.0	0.0	0.0	0.0	0.0	1.0	67.0	1.0	85.0	...	0.0	5.0	1.0	1.0

```
byte_bi_df.head()
```

	1d 6b	2e dc	cb 26	d2 86	f3 67	66 8c	8a 46	4e 47	e2 1f	49 4e	...	c3 8b	c4 0c	3b 38	3a 5c	3a dc	f0 8b
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	3.0	0.0	0.0	0.0	0.0
1	3.0	3.0	5.0	1.0	1.0	8.0	2.0	287.0	1.0	282.0	...	5.0	11.0	1.0	1.0	1.0	7.0
2	0.0	0.0	2.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	...	1.0	2.0	2.0	0.0	0.0	3.0
3	0.0	0.0	0.0	0.0	0.0	4.0	2.0	2.0	0.0	2.0	...	16.0	8.0	2.0	3.0	1.0	31.0
4	0.0	0.0	0.0	1.0	2.0	0.0	0.0	2.0	1.0	5.0	...	6.0	3.0	0.0	0.0	0.0	6.0

5 rows × 300 columns

```
result_x·=·byte_bi_df
result_y·=·data_y
result_x.tail()
```

	1d 6b	2e dc	cb 26	d2 86	f3 67	66 8c	8a 46	4e 47	e2 1f	49 4e	...	c3 8b	c4 0c	3b 38	3a 5c
10863	1.0	1.0	1.0	2.0	0.0	2.0	0.0	3.0	0.0	0.0	...	3.0	5.0	1.0	1.0
10864	2.0	1.0	0.0	1.0	0.0	18.0	41.0	56.0	1.0	56.0	...	121.0	125.0	0.0	12.0
10865	2.0	2.0	1.0	1.0	3.0	14.0	0.0	29.0	0.0	42.0	...	75.0	114.0	6.0	4.0
10866	0.0	0.0	0.0	0.0	0.0	18.0	49.0	1.0	0.0	2.0	...	131.0	48.0	0.0	0.0
10867	0.0	0.0	0.0	0.0	0.0	0.0	1.0	67.0	1.0	85.0	...	0.0	5.0	1.0	1.0

5 rows × 300 columns

```
result_y.head()
```

```
0    7
1    1
2    6
3    8
4    9
Name: Class, dtype: int64
```

1.2 Prepare, Merge and Split data

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratif
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,s
```

```
X_train.shape, y_train.shape
```

```
((8694, 300), (8694,))
```

```
result_x.shape
```

```
(10868, 300)
```

1.3 Random Forest Classifier on Bigram of ByteFiles

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------

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=1

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()
```

```
log_loss for c =  10 is 0.08367721126639815
log_loss for c =  50 is 0.06431163796062295
log_loss for c =  100 is 0.06305145455650585
log_loss for c =  500 is 0.06238874632994824
```

```
%%time
plt.close()
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  500 The train log loss is: 0.017128973414428504
For values of best alpha =  500 The cross validation log loss is: 0.06238874632994824
For values of best alpha =  500 The test log loss is: 0.049864960057818736
CPU times: user 1min 4s, sys: 2.7 s, total: 1min 7s
Wall time: 14 s
```

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

```
Number of misclassified points  1.0579576816927323
-------------------------------------------------- Confusion matrix -----------------
```

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

Predicted Class

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

1.4 Xgboost Classifier on Bigram of ByteFiles

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])  Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

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=1

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()
```

```
log_loss for c =  10 is 0.1340123554208798
log_loss for c =  50 is 0.06802316557133047
log_loss for c =  100 is 0.04884714304250573
log_loss for c =  500 is 0.04210767064887841
log_loss for c =  1000 is 0.04168808734618218
log_loss for c =  2000 is 0.04095815860110106
log_loss for c =  3000 is 0.040961270231707134
```

```
CPU times: user 1h 10min 39s, sys: 6.68 s, total: 1h 10min 46s
Wall time: 1h 10min 45s
```

```
%%time
# plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  2000 The train log loss is: 0.015158548235132147
For values of best alpha =  2000 The cross validation log loss is: 0.040958158601101
For values of best alpha =  2000 The test log loss is: 0.05835637248527767
CPU times: user 23min 17s, sys: 5.85 s, total: 23min 23s
Wall time: 23min 23s
```

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

```
Number of misclassified points  1.0579576816927323
-------------------------------------------------- Confusion matrix -----------------
-------------------------------------------------- Precision matrix -----------------
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix --------------------
```

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

1.5 Xgboost Classifier (Best Hyper Parameter) on Bigram of ByteFiles

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-wit
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)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  6.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  7.8min remaining:  4.5min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  9.3min remaining:  2.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 11.0min remaining:  1.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 15.6min finished
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', 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=False, scoring=None, verbose=10)
```

```
print(random_cfl.best_params_)
```

```
{'subsample': 0.3, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.03, 'cols
```

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------
x_cfl=XGBClassifier(n_estimators=2000,max_depth=3,learning_rate=0.03,colsample_bytree=0.3,
```

```
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)
```

```
CPU times: user 12min 5s, sys: 1.44 s, total: 12min 7s
Wall time: 12min 7s
CalibratedClassifierCV(base_estimator=XGBClassifier(base_score=0.5,
                                                    booster='gbtree',
                                                    colsample_bylevel=1,
                                                    colsample_bynode=1,
                                                    colsample_bytree=0.3,
                                                    gamma=0, learning_rate=0.03,
                                                    max_delta_step=0,
                                                    max_depth=3,
                                                    min_child_weight=1,
                                                    missing=None,
                                                    n_estimators=2000, n_jobs=1,
                                                    nthread=-1,
                                                    objective='multi:softprob',
                                                    random_state=0, reg_alpha=0,
                                                    reg_lambda=1,
                                                    scale_pos_weight=1,
                                                    seed=None, silent=None,
                                                    subsample=0.3,
                                                    verbosity=1),
                                cv='warn', method='sigmoid')
```

```
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  2000 The train log loss is: 0.01423811620716136
For values of best alpha =  2000 The cross validation log loss is: 0.041106123377502
For values of best alpha =  2000 The test log loss is: 0.05387802812381134
```

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

```
Number of misclassified points  0.8279668813247469
-------------------------------------------------- Confusion matrix -----------------
```

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

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

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names  = ['Model', 'Best Hyper Parameter', 'Train Log Loss', 'CV Log Loss', 'T

table.add_row(['Random Forest Classifier', 500, 0.017128973414428504, 0.062388746329948244
table.add_row(['XgBoost Classifier', 2000, 0.015158548235132147, 0.04095815860110106, 0.05
table.add_row(['XgBoost Classifier with Best Hyper Parameter', 2000, 0.01423811620716136,


print(table)
```

```
+---------------------------------------------+----------------------+--------------
|                    Model                    | Best Hyper Parameter |    Train Log
+---------------------------------------------+----------------------+--------------
|           Random Forest Classifier          |         500          | 0.01712897341
|              XgBoost Classifier             |         2000         | 0.0151585482
| XgBoost Classifier with Best Hyper Parameter |         2000         | 0.01423811626
+---------------------------------------------+----------------------+--------------
```

Assignment Part 2

- Watch the video (video) and include pixel intensity features to improve the logloss

2.1 Image Based Features Extraction from asm files

```
from multiprocessing import Pool
import os
from csv import writer
import numpy as np
import math
import scipy.misc
```

```
import array
import time as tm

import numpy as np
import scipy as sp
import pandas as pd
import sklearn as skl
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.feature_selection import chi2
from sklearn.metrics import log_loss, confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.model_selection import cross_val_score, KFold


def entropy(p,n):
    '''
    Calculate entropy of the file
    '''
    p_ratio = float(p)/(p+n)
    n_ratio = float(n)/(p+n)
    return -p_ratio*math.log(p_ratio) - n_ratio * math.log(n_ratio)


def info_gain(p0,n0,p1,n1,p,n):
    '''
    Calculate information gain
    '''
    return entropy(p,n) - float(p0+n0)/(p+n)*entropy(p0,n0) - float(p1+n1)/(p+n)*entropy(p


def read_image(filename):
    '''
    Read image data
    '''
    f = open(filename,'rb')
    ln = os.path.getsize(filename) # length of file in bytes
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
    a.fromfile(f,ln-rem)
    f.close()
    g = np.reshape(a,(int(len(a)/width), width))
    g = np.uint8(g)
    g = np.resize(g, (1000,))
    return list(g)


def extract_asm_image_features(tfiles):
    '''
    Extract image features from the asm files
    '''
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    # Generate feature file csv
```

```
    pid = os.getpid()
    feature_file = str(pid) + '-image-features-asm.csv'

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in range(1000)]
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image('asmFiles/'+ fname)
            outrows.append([file_id] + image_data)

            # Print progress
            if (idx+1) % 100 == 0:
                print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

        # Write remaining files
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []


# Now divide the train files into four groups for multiprocessing
start_time = tm.time()
tfiles = os.listdir('asmFiles')
quart = int(len(tfiles)/4)
# print(quart)
train1 = tfiles[:quart]
train2 = tfiles[quart:(2*quart)]
train3 = tfiles[(2*quart):(3*quart)]
train4 = tfiles[(3*quart):]
print(len(tfiles), quart, (len(train1)+len(train2)+len(train3)+len(train4)))
trains = [train1, train2, train3, train4]
p = Pool(4)
p.map(extract_asm_image_features, trains)
print("Elapsed time: {:.2f} hours.".format((tm.time() - start_time)/3600.0))
```

```
2320 1700 of 2717 files processed.
2319 1800 of 2717 files processed.
2318 1700 of 2717 files processed.
2317 1600 of 2717 files processed.
2319 1900 of 2717 files processed.
2320 1800 of 2717 files processed.
2318 1800 of 2717 files processed.
2317 1700 of 2717 files processed.
2319 2000 of 2717 files processed.
2320 1900 of 2717 files processed.
2317 1800 of 2717 files processed.
2318 1900 of 2717 files processed.
2319 2100 of 2717 files processed.
2320 2000 of 2717 files processed.
2318 2000 of 2717 files processed.
2317 1900 of 2717 files processed.
2319 2200 of 2717 files processed.
```

```
2320 2100 of 2717 files processed.
2317 2000 of 2717 files processed.
2318 2100 of 2717 files processed.
2319 2300 of 2717 files processed.
2320 2200 of 2717 files processed.
2317 2100 of 2717 files processed.
2318 2200 of 2717 files processed.
2319 2400 of 2717 files processed.
2320 2300 of 2717 files processed.
2317 2200 of 2717 files processed.
2318 2300 of 2717 files processed.
2319 2500 of 2717 files processed.
2320 2400 of 2717 files processed.
2317 2300 of 2717 files processed.
2318 2400 of 2717 files processed.
2320 2500 of 2717 files processed.
2319 2600 of 2717 files processed.
2318 2500 of 2717 files processed.
2317 2400 of 2717 files processed.
2319 2700 of 2717 files processed.
2320 2600 of 2717 files processed.
2318 2600 of 2717 files processed.
2317 2500 of 2717 files processed.
2320 2700 of 2717 files processed.
2318 2700 of 2717 files processed.
2317 2600 of 2717 files processed.
2317 2700 of 2717 files processed.
Elapsed time: 0.39 hours.
Process ForkPoolWorker-13:
Traceback (most recent call last):
  File "/home/mayankgupta/anaconda3/lib/python3.7/multiprocessing/process.py", lin
    self.run()
  File "/home/mayankgupta/anaconda3/lib/python3.7/multiprocessing/process.py", lin
    self._target(*self._args, **self._kwargs)
  File "/home/mayankgupta/anaconda3/lib/python3.7/multiprocessing/pool.py", line 1
    task = get()
  File "/home/mayankgupta/anaconda3/lib/python3.7/multiprocessing/queues.py", line
    with self._rlock:
  File "/home/mayankgupta/anaconda3/lib/python3.7/multiprocessing/synchronize.py",
    return self._semlock.__enter__()
```

```
#merging all generated csv files

labels = pd.read_csv('trainLabels.csv')
d1 = pd.read_csv('2317-image-features-asm.csv')
d2 = pd.read_csv('2318-image-features-asm.csv')
d3 = pd.read_csv('2319-image-features-asm.csv')
d4 = pd.read_csv('2320-image-features-asm.csv')
d4.shape
```

```
(2717, 1001)
```

```
data = pd.concat([d1, d2, d3, d4])
data.shape
```

```
(10868, 1001)
```

```
data.reset_index(drop=True, inplace=True)
```

```
labels.head()
```

	Id	Class
0	01kcPWA9K2BOxQeS5Rju	1
1	04EjIdbPV5e1XroFOpiN	1
2	05EeG39MTRrI6VY21DPd	1
3	05rJTUWYAKNegBk2wE8X	1
4	0AnoOZDNbPXIr2MRBSCJ	1

```
sorted_train_data = data.sort_values(by='filename', axis=0, ascending=True, inplace=False)
sorted_train_labels = labels.sort_values(by='Id', axis=0, ascending=True, inplace=False)
X = sorted_train_data.iloc[:,1:]
y = np.array(sorted_train_labels.iloc[:,1])
```

```
X.shape, y.shape
```

```
((10868, 1000), (10868,))
```

2.1.1 Selecting top 50% variance features

```
# find the top 50 percent variance features, from 1000 -> 500 features
fsp = SelectPercentile(chi2, 50)
X_new_50 = fsp.fit_transform(X,y)
X_new_50.shape
```

```
(10868, 500)
```

```
selected_names = fsp.get_support(indices=True)
selected_names = selected_names + 1
selected_names
```

```
array([  2,   4,   5,  15,  21,  22,  24,  25,  26,  27,  29,  30,  32,
        33,  34,  35,  41,  42,  43,  44,  48,  50, 125, 126, 135, 136,
       138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 151, 152,
       154, 155, 156, 157, 158, 160, 161, 162, 163, 164, 165, 167, 169,
       173, 174, 179, 186, 188, 190, 198, 201, 202, 205, 215, 216, 217,
       219, 220, 221, 222, 223, 224, 226, 227, 229, 236, 240, 241, 242,
       243, 244, 245, 246, 247, 248, 249, 252, 253, 260, 261, 262, 263,
       264, 265, 266, 267, 268, 269, 271, 272, 273, 282, 287, 291, 292,
       293, 294, 295, 296, 297, 307, 308, 310, 311, 312, 313, 314, 315,
```

```
       316, 317, 318, 319, 321, 323, 326, 327, 328, 330, 334, 337, 338,
       339, 340, 341, 343, 344, 345, 346, 349, 350, 351, 352, 353, 354,
       356, 357, 358, 359, 366, 367, 368, 370, 371, 372, 373, 374, 375,
       376, 378, 379, 380, 381, 384, 385, 386, 387, 388, 390, 391, 392,
       399, 400, 401, 402, 403, 404, 405, 408, 409, 410, 412, 413, 414,
       415, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431,
       436, 437, 439, 440, 441, 442, 443, 445, 446, 447, 448, 449, 450,
       451, 452, 453, 457, 458, 459, 460, 461, 464, 465, 466, 467, 477,
       478, 479, 480, 481, 482, 538, 539, 555, 556, 557, 558, 559, 560,
       561, 563, 564, 567, 568, 571, 572, 573, 580, 581, 582, 583, 584,
       585, 586, 587, 588, 589, 590, 597, 598, 600, 601, 602, 603, 606,
       607, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624,
       627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 640, 641,
       642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654,
       655, 656, 657, 658, 659, 662, 664, 670, 671, 672, 673, 674, 675,
       676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688,
       689, 691, 692, 693, 694, 695, 696, 701, 702, 703, 704, 708, 709,
       711, 712, 713, 714, 715, 717, 718, 719, 720, 721, 722, 723, 724,
       725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 738,
       739, 740, 743, 744, 751, 752, 753, 754, 755, 756, 757, 758, 759,
       760, 761, 762, 763, 765, 774, 775, 776, 777, 778, 779, 780, 781,
       782, 784, 785, 786, 787, 788, 789, 793, 798, 801, 802, 813, 814,
       818, 819, 820, 830, 831, 835, 836, 837, 838, 840, 841, 847, 848,
       849, 850, 851, 852, 853, 855, 856, 857, 866, 867, 868, 869, 870,
       873, 874, 875, 876, 877, 878, 879, 882, 898, 899, 904, 907, 908,
       919, 920, 923, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939,
       940, 941, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957,
       958, 959, 960, 961, 962, 963, 965, 966, 967, 968, 973, 974, 975,
       976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 989, 990, 991,
       992, 995, 996, 997, 998, 999])
```

```
data_trimmed = sorted_train_data.iloc[:,selected_names]
data_fnames = pd.DataFrame(sorted_train_data['filename'])
data_reduced = data_fnames.join(data_trimmed)
data_reduced.head()
```

	filename	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23
7551	01IsoiSMh5gxyDYTl4CB	116	120	116	9	32	32	32
6521	01SuzwMJEIXsK7A8dQbl	69	68	69	48	9	9	13
10798	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13
610	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13
6715	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13

5 rows × 501 columns

```
data_reduced.to_csv('sorted-features-asm-50percent.csv', index=False)
```

```
data_reduced = pd.read_csv('sorted-features-asm-50percent.csv')
data_reduced.shape
```

```
(10868, 501)
```

```
data_reduced.rename(columns={'filename': 'ID'}, inplace=True)
```

```
data_reduced.shape
```

```
(10868, 501)
```

```
data_reduced.head()
```

	ID	ASM_1	ASM_3	ASM_4	ASM_14	ASM_20	ASM_21	ASM_23	ASM_
0	01IsoiSMh5gxyDYTl4CB	116	120	116	9	32	32	32	
1	01SuzwMJEIXsK7A8dQbl	69	68	69	48	9	9	13	
2	01azqd4InC7m9JpocGv5	69	68	69	48	9	9	13	
3	01jsnpXSAlgw6aPeDxrU	69	68	69	48	9	9	13	
4	01kcPWA9K2BOxQeS5Rju	69	68	69	48	9	9	13	

5 rows × 501 columns

2.2 Implement asm image features + bytes uni-gram features

2.2.1 Merge asm image features + bytes uni-gram features

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

	0	1	2	3	4	5	6	7	
0	0.002525	0.000082	0.000013	0.000017	0.000016	0.000012	0.000004	0.000003	0.000
1	0.010740	0.001771	0.000416	0.000489	0.000886	0.000320	0.000332	0.000662	0.000
2	0.005374	0.000624	0.000130	0.000249	0.000129	0.000100	0.000105	0.000168	0.000
3	0.008818	0.000957	0.000176	0.000247	0.000174	0.000207	0.000123	0.000221	0.000
4	0.037465	0.000991	0.000251	0.000246	0.000315	0.000366	0.000252	0.000369	0.000

5 rows × 758 columns

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratif
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,s
```

2.2.2 Random Forest Classifier asm image features + bytes uni-gram features

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------

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=1

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()
```

```
log_loss for c =  10 is 0.03298443094752871
log_loss for c =  50 is 0.03085568636919456
log_loss for c =  100 is 0.030007763918641725
log_loss for c =  500 is 0.02966720993128468
log_loss for c =  1000 is 0.029971997588127103
log_loss for c =  2000 is 0.030181377953265455
log_loss for c =  3000 is 0.0301742816771141
```

```
CPU times: user 18min 20s, sys: 32 s, total: 18min 52s
Wall time: 3min 11s
```

2.2.3 Random Forest Classifier (Best Hypher Parameters) asm image features + bytes uni-gram features

```
%%time
plt.close()
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  500 The train log loss is: 0.012780736170043525
For values of best alpha =  500 The cross validation log loss is: 0.02966720993128468
For values of best alpha =  500 The test log loss is: 0.034478799054374044
CPU times: user 1min 27s, sys: 4.1 s, total: 1min 31s
Wall time: 16.7 s
```

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

```
Number of misclassified points  0.6439742410303588
-------------------------------------------------- Confusion matrix -----------------
```

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

2.2.4 Xgboost Classifier asm image features + bytes uni-gram features

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
```

```
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

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=1

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()
```

```
log_loss for c =  10 is 0.0792497193661065
log_loss for c =  50 is 0.031980906132399546
log_loss for c =  100 is 0.025713754198451427
log_loss for c =  500 is 0.024323002938319807
log_loss for c =  1000 is 0.02476490339663419
log_loss for c =  2000 is 0.02516493136273984
log_loss for c =  3000 is 0.025065229722220864
```

```
%%time
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  500 The train log loss is: 0.01098127054981705
For values of best alpha =  500 The cross validation log loss is: 0.0250652297222208
For values of best alpha =  500 The test log loss is: 0.033166108866613246
CPU times: user 1h 18min 17s, sys: 0 ns, total: 1h 18min 17s
Wall time: 1h 18min 17s
```

2.2.5 Xgboost Classifier (Best Hyper Parameters) asm image features + bytes uni-gram features

```
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)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  6.5min remaining:  3.7min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 11.3min remaining:  3.4min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 16.0min finished
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', 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=False, scoring=None, verbose=10)
```

```
print (random_cfl.best_params_)
```

```
{'subsample': 0.3, 'n_estimators': 1000, 'max_depth': 3, 'learning_rate': 0.15, 'cols
```

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------
```

```
x_cfl=XGBClassifier(n_estimators=1000,max_depth=3,learning_rate=0.15,colsample_bytree=0.3,
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  500 The train log loss is: 0.011284904939918825
For values of best alpha =  500 The cross validation log loss is: 0.0245278997147241
For values of best alpha =  500 The test log loss is: 0.028035316596927817
CPU times: user 10min 40s, sys: 192 ms, total: 10min 40s
Wall time: 10min 40s
```

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

```
Number of misclassified points  0.45998160073597055
-------------------------------------------------- Confusion matrix -----------------
```

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

2.2.6 Conclusion and Model Comparision (asm image features + bytes unigram features)

```
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV Log Loss', 'Te

table.add_row(['Random Forest Classifier', 500, 0.012780736170043525, 0.02966720993128468,
table.add_row(['Xgboost Classifier', 500, 0.011284904939918825, 0.024527899714724164, 0.02

print(table)
```

```
+-------------------------+---------------------+----------------------+------------
|          Model          | Best HyperParameter |    Train Log Loss    |     CV Log
+-------------------------+---------------------+----------------------+------------
| Random Forest Classifier |         500         | 0.012780736170043525 | 0.029667209
|    Xgboost Classifier    |         500         | 0.011284904939918825 | 0.024527899
+-------------------------+---------------------+----------------------+------------
```

2.3 Implement asm unigram + asm extracted image features

2.3.1 Merge asm unigram + asm extracted image features

```
print(data_reduced.shape)
print(result_asm.shape)
```

```
(10868, 501)
(10868, 54)
```

```
result_asm.columns
```

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

```
data_reduced.columns
```

```
Index(['ID', 'ASM_1', 'ASM_3', 'ASM_4', 'ASM_14', 'ASM_20', 'ASM_21', 'ASM_23',
       'ASM_24', 'ASM_25',
       ...
       'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_994',
       'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
      dtype='object', length=501)
```

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

t:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	.tls:	...	ASM_984	A
92	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	0.0	...	83	
30	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	0.0	...	114	
27	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	0.0	...	32	
33	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	0.0	...	114	
90	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	0.0	...	114	

```
result_x.columns
```

```
Index(['HEADER:', '.text:', '.Pav:', '.idata:', '.data:', '.bss:', '.rdata:',
       '.edata:', '.rsrc:', '.tls:',
       ...
       'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_994',
       'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
      dtype='object', length=549)
```

```
print(result_x.shape)
```

```
(10868, 549)
```

```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratif
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,s
```

2.3.2 Random Forest Classifier with asm unigram + asm extracted image features

```
%%time
plt.close()
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------

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=1

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()
```

```
log_loss for c =  10 is 0.021147757013626152
log_loss for c =  50 is 0.019795586264623286
log_loss for c =  100 is 0.019531514435224526
log_loss for c =  500 is 0.019132586421543085
log_loss for c =  1000 is 0.01893597948023988
log_loss for c =  2000 is 0.018992036016380148
log_loss for c =  3000 is 0.01906281256167374
```

```
%%time
plt.close()
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  1000 The train log loss is: 0.008896043075915374
For values of best alpha =  1000 The cross validation log loss is: 0.018935979480239
For values of best alpha =  1000 The test log loss is: 0.02220239178075276
CPU times: user 1min 16s, sys: 6.4 s, total: 1min 22s
Wall time: 22.9 s
```

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

```
Number of misclassified points  0.3219871205151794
-------------------------------------------------- Confusion matrix -----------------
```

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

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

2.3.3 Xgboost implementation with asm unigram + asm extracted image features

```
%%time
plt.close()
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

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=1

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()
```

```
log_loss for c =  10 is 0.06278899176536516
log_loss for c =  50 is 0.035408785513269654
log_loss for c =  100 is 0.02871795575938164
log_loss for c =  500 is 0.027998979315273183
log_loss for c =  1000 is 0.027561393234936435
log_loss for c =  2000 is 0.02725772002665156
log_loss for c =  3000 is 0.02725750007288518
```

```
CPU times: user 1h 48min 55s, sys: 363 ms, total: 1h 48min 55s
Wall time: 1h 48min 55s
```

```
%%time
```

```
plt.close()
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  3000 The train log loss is: 0.008320609361993012
For values of best alpha =  3000 The cross validation log loss is: 0.0272575000728851
For values of best alpha =  3000 The test log loss is: 0.02159222942854402
CPU times: user 45min, sys: 67.8 ms, total: 45min
Wall time: 45min
```

2.3.4 Xgboost implementation (Best HyperParameter) with asm unigram + asm extracted image features

```
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)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   36.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed: 11.5min remaining:  6.6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 12.8min remaining:  3.9min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 22.9min finished
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
```

```
                                        seed=None, silent=None, subsample=1,
                                        verbosity=1),
                        iid='warn', 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=False, scoring=None, verbose=10)
```

```
print (random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.2, 'colsam
```

```
%%time
# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/pyth
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=Tru
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_chil
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg
# 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=Non
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functi
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/wha
# -----------------------

x_cfl=XGBClassifier(n_estimators=500,max_depth=3,learning_rate=0.2,colsample_bytree=0.5,su
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
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
For values of best alpha =  3000 The train log loss is: 0.007914584198355933
For values of best alpha =  3000 The cross validation log loss is: 0.0272626406859971
For values of best alpha =  3000 The test log loss is: 0.02043044592140415
CPU times: user 6min 33s, sys: 28 ms, total: 6min 33s
Wall time: 6min 33s
```

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

```
Number of misclassified points  0.18399264029438822
-------------------------------------------------- Confusion matrix ----------------
```

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

2.3.5 Conclusion and Model Comparision (asm unigram + asm extracted image features)

```
table = PrettyTable()
table.field_names  = ['Model', 'Best HyperParameter', 'Train Log Loss', 'CV Log Loss', 'Te

table.add_row(['Random Forest Classifier', 1000, 0.008896043075915374, 0.01893597948023988
table.add_row(['Xgboost Classifier', 3000, 0.007914584198355933, 0.027262640685997187, 0.0

print(table)
```

```
+-------------------------+---------------------+----------------------+------------
|          Model          | Best HyperParameter |    Train Log Loss    |     CV Log
+-------------------------+---------------------+----------------------+------------
| Random Forest Classifier |         1000        | 0.008896043075915374 | 0.018935979
|    Xgboost Classifier   |         3000        | 0.007914584198355933 | 0.027262640
+-------------------------+---------------------+----------------------+------------
```

2.4 Implemented ASM unigram + ASM image features + ByteFile unigram

2.4.1 Prepare, Merge and Split (ASM unigram + ASM image features + ByteFile unigram) data

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

```
(10868, 501)
(10868, 54)
(10868, 261)
```

```
data_reduced.columns
```

```
Index(['ID', 'ASM_1', 'ASM_3', 'ASM_4', 'ASM_14', 'ASM_20', 'ASM_21', 'ASM_23',
       'ASM_24', 'ASM_25',
       ...
       'ASM_984', 'ASM_988', 'ASM_989', 'ASM_990', 'ASM_991', 'ASM_994',
       'ASM_995', 'ASM_996', 'ASM_997', 'ASM_998'],
      dtype='object', length=501)
```

```
result_asm.columns
```

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

```
result.columns
```

```
Index(['ID', '0', '1', '2', '3', '4', '5', '6', '7', '8',
       ...
       'fa', 'fb', 'fc', 'fd', 'fe', 'ff', '??', 'Unnamed: 0', 'size',
       'Class'],
      dtype='object', length=261)
```

```
result_y
```

```
0        1
1        1
2        1
3        1
```