4/6/2021 temp-161770208870318078

**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 (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**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 1/159

4/6/2021 temp-161770208870318078

**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**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 2/159

4/6/2021 temp-161770208870318078

**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**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 3/159

4/6/2021 temp-161770208870318078

**.asm file**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 4/159

4/6/2021 temp-161770208870318078

**.text:00401000 assume es:nothing, ss:nothing, ds:\_data, fs:nothing, gs:nothing .text:00401000 56 push esi**

**.text:00401001 8D 44 24 08 lea eax, [esp+8]**

**.text:00401005 50 push eax**

**.text:00401006 8B F1 mov esi, ecx**

**.text:00401008 E8 1C 1B 00 00 call ??0exception@std@@QAE@ABQBD@Z ; std::exceptio n::exception(char const \* const &)**

**.text:0040100D C7 06 08 BB 42 00 mov dword ptr [esi], offset off\_42BB08 .text:00401013 8B C6 mov eax, esi**

**.text:00401015 5E pop esi**

**.text:00401016 C2 04 00 retn 4**

**.text:00401016 ; --------------------------------------------------------------------- ------**

**.text:00401019 CC CC CC CC CC CC CC align 10h**

**.text:00401020 C7 01 08 BB 42 00 mov dword ptr [ecx], offset off\_42BB08 .text:00401026 E9 26 1C 00 00 jmp sub\_402C51**

**.text:00401026 ; --------------------------------------------------------------------- ------**

**.text:0040102B CC CC CC CC CC align 10h**

**.text:00401030 56 push esi**

**.text:00401031 8B F1 mov esi, ecx**

**.text:00401033 C7 06 08 BB 42 00 mov dword ptr [esi], offset off\_42BB08 .text:00401039 E8 13 1C 00 00 call sub\_402C51**

**.text:0040103E F6 44 24 08 01 test byte ptr [esp+8], 1**

**.text:00401043 74 09 jz short loc\_40104E**

**.text:00401045 56 push esi**

**.text:00401046 E8 6C 1E 00 00 call ??3@YAXPAX@Z ; operator delete(void \*) .text:0040104B 83 C4 04 add esp, 4**

**.text:0040104E**

**.text:0040104E loc\_40104E: ; CODE XREF: .text:00401043j .text:0040104E 8B C6 mov eax, esi**

**.text:00401050 5E pop esi**

**.text:00401051 C2 04 00 retn 4**

**.text:00401051 ; --------------------------------------------------------------------- ------**

**.bytes file**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 5/159

4/6/2021 temp-161770208870318078

**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**

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

**2.2.1. Type of Machine Learning Problem**

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

**2.2.2. Performance Metric**

**Source: https://www.kaggle.com/c/malware-classification#evaluation (https://www.kaggle.com/c/malware-classification#evaluation) Metric(s):**

**Multi class log-loss**

**Confusion matrix**

**2.2.3. Machine Learing Objectives and Constraints**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 6/159

4/6/2021 temp-161770208870318078

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

**Class probabilities are needed.**

**Penalize the errors in class probabilites => Metric is Log-loss.**

**Some Latency constraints.**

**2.3. Train and Test Dataset**

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

**2.4. Useful blogs, videos and reference papers**

**http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/ https://arxiv.org/pdf/1511.04317.pdf**

**First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y https://github.com/dchad/malware-detection**

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

**https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu\_pIB6ua?dl=0**

**" Cross validation is more trustworthy than domain knowledge."**

**3. Exploratory Data Analysis**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 7/159

4/6/2021 temp-161770208870318078

**In [ ]: import warnings**

**warnings.filterwarnings("ignore")**

**import shutil**

**from joblib import Parallel, delayed**

**import os**

***# os.environ["MODIN\_ENGINE"] = "dask"***

***# import modin.pandas as pd***

**import pandas as pd**

**import matplotlib**

**matplotlib.use(u'nbAgg')**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import numpy as np**

**import array**

**import csv**

**from csv import writer,reader**

**import pickle**

**from sklearn.manifold import TSNE**

**from sklearn import preprocessing**

**from multiprocessing import Process*# this is used for multithreading* from multiprocessing import Pool**

**import multiprocessing**

**import codecs*# this is used for file operations***

**import random as r**

**from tqdm.notebook import tqdm**

**import scipy**

**from sklearn.feature\_extraction.text import CountVectorizer 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**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 8/159

4/6/2021 temp-161770208870318078

**In [ ]: *#separating byte files and asm files***

**source = 'train'**

**destination\_1 = 'byteFiles'**

**destination\_2 = 'asmFiles'**

***# we will check if the folder 'byteFiles' exists if it not there we will create a folder with the same name* if not os.path.isdir(destination\_1):**

**os.makedirs(destination\_1)**

**if not os.path.isdir(destination\_2):**

**os.makedirs(destination\_2)**

***# if we have folder called 'train' (train folder contains both .asm files and .bytes files) we will rename it 'asmFiles'***

***# for every file that we have in our 'asmFiles' directory we check if it is ending with .bytes, if yes we wil l move it to***

***# 'byteFiles' folder***

***# so by the end of this snippet we will separate all the .byte files and .asm files***

**if os.path.isdir(source):**

**data\_files = os.listdir(source)**

**for file in data\_files:**

**print(file)**

**if (file.endswith("bytes")):**

**shutil.move(source+'\\'+file,destination\_1)**

**if (file.endswith("asm")):**

**shutil.move(source+'\\'+file,destination\_2)**

**3.1. Distribution of malware classes in whole data set**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 9/159

4/6/2021 temp-161770208870318078

**In [ ]: Y=pd.read\_csv("trainLabels.csv")**

**total = len(Y)\*1.**

**ax=sns.countplot(x="Class", data=Y)**

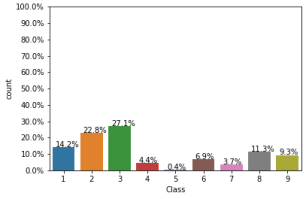
**for p in ax.patches:**

**ax.annotate('{:.1f}%'.format(100\*p.get\_height()/total), (p.get\_x()+0.1, p.get\_height()+5))**

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

***#adjust the ticklabel to the desired format, without changing the position of the ticks.* ax.set\_yticklabels(map('{:.1f}%'.format, 100\*ax.yaxis.get\_majorticklocs()/total))**

**plt.show()**

****

**3.2. Feature extraction**

**3.2.1 File size of byte files as a feature**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 10/159

4/6/2021 temp-161770208870318078

**In [ ]: *#file sizes of byte files***

**files=os.listdir('byteFiles')**

**filenames=Y['Id'].tolist()**

**class\_y=Y['Class'].tolist()**

**class\_bytes=[]**

**sizebytes=[]**

**fnames=[]**

**for file in files:**

***# print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))***

***# os.stat\_result(st\_mode=33206, st\_ino=1125899906874507, st\_dev=3561571700, st\_nlink=1, st\_uid=0, st\_gid= 0,***

***# st\_size=3680109, st\_atime=1519638522, st\_mtime=1519638522, st\_ctime=1519638522)***

***# read more about os.stat: here https://www.tutorialspoint.com/python/os\_stat.htm***

**statinfo=os.stat('byteFiles/'+file)**

***# split the file name at '.' and take the first part of it i.e the file name***

**file=file.split('.')[0]**

**if any(file == filename for filename in filenames):**

**i=filenames.index(file)**

**class\_bytes.append(class\_y[i])**

***# converting into Mb's***

**sizebytes.append(statinfo.st\_size/(1024.0\*1024.0))**

**fnames.append(file)**

**data\_size\_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class\_bytes})**

**print (data\_size\_byte.head())**

**ID size Class**

**0 01azqd4InC7m9JpocGv5 5.012695 9**

**1 01IsoiSMh5gxyDYTl4CB 6.556152 2**

**2 01jsnpXSAlgw6aPeDxrU 4.602051 9**

**3 01kcPWA9K2BOxQeS5Rju 0.679688 1**

**4 01SuzwMJEIXsK7A8dQbl 0.438965 8**

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

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 11/159

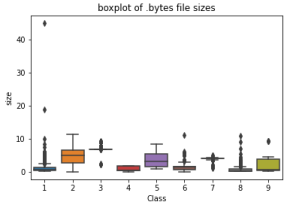
4/6/2021 temp-161770208870318078

**In [ ]: *#boxplot of byte files***

**data\_size\_byte = pd.read\_csv('result\_with\_size.csv')**

**data\_size\_byte.drop('Unnamed: 0',axis =1, inplace = True) ax = sns.boxplot(x="Class", y="size", data=data\_size\_byte) plt.title("boxplot of .bytes file sizes")**

**plt.show()**

****

**3.2.3 feature extraction from byte files**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 12/159

4/6/2021 temp-161770208870318078

**In [ ]: *#removal of addres from byte files***

***# contents of .byte files***

***# ----------------***

***#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08***

***#-------------------***

***#we remove the starting address 00401000***

**files = os.listdir('byteFiles')**

**filenames=[]**

**array=[]**

**for file in files:**

**if(file.endswith("bytes")):**

**file=file.split('.')[0]**

**text\_file = open('byteFiles/'+file+".txt", 'w+')**

**with open('byteFiles/'+file+".bytes","r") as fp:**

**lines=""**

**for line in fp:**

**a=line.rstrip().split(" ")[1:]**

**b=' '.join(a)**

**b=b+"\n"**

**text\_file.write(b)**

**fp.close()**

**os.remove('byteFiles/'+file+".bytes")**

**text\_file.close()**

**files = os.listdir('byteFiles')**

**filenames2=[]**

**feature\_matrix = np.zeros((len(files),257),dtype=int)**

**k=0**

***#program to convert into bag of words of bytefiles***

***#this is custom-built bag of words this is unigram bag of words***

**byte\_feature\_file=open('result.csv','w+')**

**byte\_feature\_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1 e,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,40,41,4 2,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,6 6,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,87,88,89,8 a,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,a e,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,d 2,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,f 6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 13/159

4/6/2021 temp-161770208870318078

**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()**

**In [ ]: byte\_features=pd.read\_csv("result.csv")**

**byte\_features['ID'] = byte\_features['ID'].str.split('.').str[0]**

**byte\_features.head(2)**

**Out[ ]:**

**ID 0 1 2 3 4 5 6 7 8 9 0a 0b 0c 0d 0e 0f 10 0 01azqd4InC7m9JpocGv5 601905 3905 2816 3832 3345 3242 3650 3201 2965 3205 3211 3546 4038 4096 3218 3032 3269 1 01IsoiSMh5gxyDYTl4CB 39755 8337 7249 7186 8663 6844 8420 7589 9291 358 340 6649 8660 447 218 6869 8869**

**2 rows × 258 columns**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 14/159

4/6/2021 temp-161770208870318078

**In [ ]: data\_size\_byte.head(2)**

**Out[ ]:**

**ID size Class**

**0 01azqd4InC7m9JpocGv5 4.234863 9**

**1 01IsoiSMh5gxyDYTl4CB 5.538818 2**

**In [ ]: byte\_features\_with\_size = data\_size\_byte*#\_features.merge(data\_size\_byte, on='ID') # byte\_features\_with\_size.to\_csv("result\_with\_size.csv",index = False)***

***# byte\_features\_with\_size.drop('Unnamed: 0',axis =1, inplace = True)***

**byte\_features\_with\_size.head(2)**

**Out[ ]:**

**ID 0 1 2 3 4 5 6 7 8 9 0a 0b 0c 0d 0e 0f 10 0 01azqd4InC7m9JpocGv5 601905 3905 2816 3832 3345 3242 3650 3201 2965 3205 3211 3546 4038 4096 3218 3032 3269 1 01IsoiSMh5gxyDYTl4CB 39755 8337 7249 7186 8663 6844 8420 7589 9291 358 340 6649 8660 447 218 6869 8869**

**2 rows × 260 columns**

**In [ ]: byte\_features\_with\_size = pd.read\_csv('result\_with\_size.csv')**

**byte\_features\_with\_size.drop(columns=["Unnamed: 0"],inplace =True)**

**byte\_features\_with\_size.head()**

**Out[ ]:**

**ID 0 1 2 3 4 5 6 7 8 9 0a 0b 0c 0d 0e 0f 1**

**0 01azqd4InC7m9JpocGv5 601905 3905 2816 3832 3345 3242 3650 3201 2965 3205 3211 3546 4038 4096 3218 3032 326 1 01IsoiSMh5gxyDYTl4CB 39755 8337 7249 7186 8663 6844 8420 7589 9291 358 340 6649 8660 447 218 6869 886 2 01jsnpXSAlgw6aPeDxrU 93506 9542 2568 2438 8925 9330 9007 2342 9107 2457 2655 2669 9113 2584 2788 2487 278 3 01kcPWA9K2BOxQeS5Rju 21091 1213 726 817 1257 625 550 523 1078 473 516 445 808 432 403 705 106 4 01SuzwMJEIXsK7A8dQbl 19764 710 302 433 559 410 262 249 422 223 237 226 406 643 213 272 44**

**5 rows × 260 columns**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 15/159

4/6/2021 temp-161770208870318078

**In [ ]: *# https://stackoverflow.com/a/29651514***

**def normalize(df):**

**result1 = df.copy()**

**for feature\_name in df.columns:**

**if (str(feature\_name) != str('ID') and str(feature\_name)!=str('Class')):**

**max\_value = df[feature\_name].max()**

**min\_value = df[feature\_name].min()**

**result1[feature\_name] = (df[feature\_name] - min\_value) / (max\_value - min\_value) return result1**

**result = normalize(byte\_features\_with\_size)**

**---------------------------------------------------------------------------**

**NameError Traceback (most recent call last)**

**<ipython-input-2-029acda11a5e> in <module>()**

**8 result1[feature\_name] = (df[feature\_name] - min\_value) / (max\_value - min\_value) 9 return result1**

**---> 10 result = normalize(byte\_features\_with\_size)**

**NameError: name 'byte\_features\_with\_size' is not defined**

**In [ ]: result.head(2)**

**Out[ ]:**

**ID 0 1 2 3 4 5 6 7 8 9**

**0 01azqd4InC7m9JpocGv5 0.262806 0.005498 0.001567 0.002067 0.002048 0.001835 0.002058 0.002946 0.002638 0.003531 0.0068 1 01IsoiSMh5gxyDYTl4CB 0.017358 0.011737 0.004033 0.003876 0.005303 0.003873 0.004747 0.006984 0.008267 0.000394 0.0007**

**2 rows × 260 columns**

**In [ ]: data\_y = result['Class']**

**data\_y.head()**

**Out[ ]: 0 9**

**1 2**

**2 9**

**3 1**

**4 8**

**Name: Class, dtype: int64**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 16/159

4/6/2021 temp-161770208870318078

**3.2.4 Multivariate Analysis**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 17/159

4/6/2021 temp-161770208870318078

**In [ ]: *#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]**

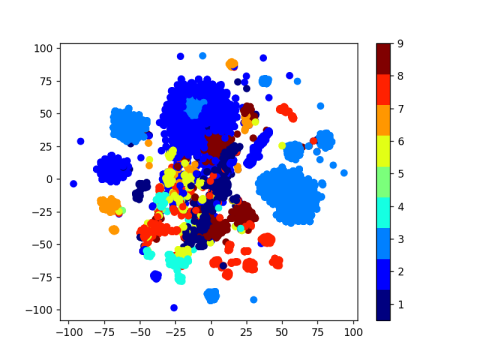
**vis\_y = results[:, 1]**

**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()**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 18/159

4/6/2021 temp-161770208870318078

**In [ ]: *#this is with perplexity 30***

**xtsne=TSNE(perplexity=30)**

**results=xtsne.fit\_transform(result.drop(['ID','Class'], axis=1))**

**vis\_x = results[:, 0]**

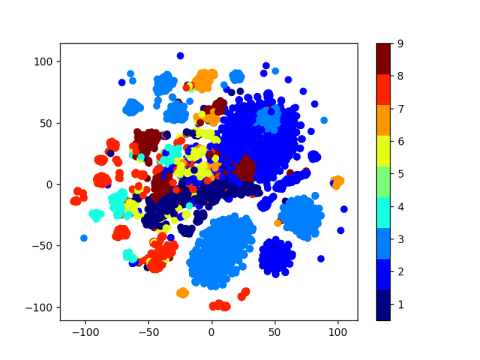
**vis\_y = results[:, 1]**

**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()**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 19/159

4/6/2021 temp-161770208870318078

**Train Test split**

**In [ ]: data\_y = result['Class']**

***# split the data into test and train by maintaining same distribution of output varaible 'y\_true' [stratify=y \_true]***

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(result.drop(['ID','Class'], axis=1), data\_y,stratify=data \_y,test\_size=0.20)**

***# split the train data into train and cross validation by maintaining same distribution of output varaible 'y \_train' [stratify=y\_train]***

**X\_train, X\_cv, y\_train, y\_cv = train\_test\_split(X\_train, y\_train,stratify=y\_train,test\_size=0.20)**

**In [ ]: 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**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 20/159

4/6/2021 temp-161770208870318078

**In [ ]: *# it returns a dict, keys as class labels and values as the number of data points in that class* train\_class\_distribution = y\_train.value\_counts().sortlevel()**

**test\_class\_distribution = y\_test.value\_counts().sortlevel()**

**cv\_class\_distribution = y\_cv.value\_counts().sortlevel()**

**my\_colors = 'rgbkymc'**

**train\_class\_distribution.plot(kind='bar', color=my\_colors)**

**plt.xlabel('Class')**

**plt.ylabel('Data points per Class')**

**plt.title('Distribution of yi in train data')**

**plt.grid()**

**plt.show()**

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

***# -(train\_class\_distribution.values): the minus sign will give us in decreasing order***

**sorted\_yi = np.argsort(-train\_class\_distribution.values)**

**for i in sorted\_yi:**

**print('Number of data points in class', i+1, ':',train\_class\_distribution.values[i], '(', np.round((train \_class\_distribution.values[i]/y\_train.shape[0]\*100), 3), '%)')**

**print('-'\*80)**

**my\_colors = 'rgbkymc'**

**test\_class\_distribution.plot(kind='bar', color=my\_colors)**

**plt.xlabel('Class')**

**plt.ylabel('Data points per Class')**

**plt.title('Distribution of yi in test data')**

**plt.grid()**

**plt.show()**

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

***# -(train\_class\_distribution.values): the minus sign will give us in decreasing order***

**sorted\_yi = np.argsort(-test\_class\_distribution.values)**

**for i in sorted\_yi:**

**print('Number of data points in class', i+1, ':',test\_class\_distribution.values[i], '(', np.round((test\_c lass\_distribution.values[i]/y\_test.shape[0]\*100), 3), '%)')**

**print('-'\*80)**

**my\_colors = 'rgbkymc'**

**cv\_class\_distribution.plot(kind='bar', color=my\_colors)**

**plt.xlabel('Class')**

**plt.ylabel('Data points per Class')**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 21/159

4/6/2021 temp-161770208870318078

**plt.title('Distribution of yi in cross validation data')**

**plt.grid()**

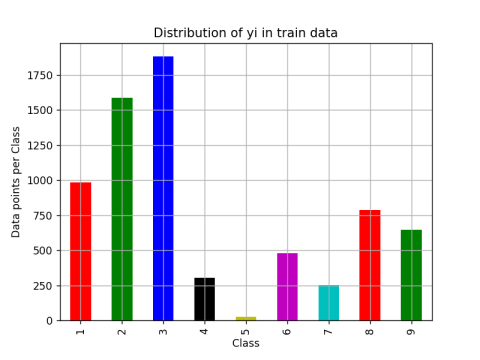
**plt.show()**

***# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html # -(train\_class\_distribution.values): the minus sign will give us in decreasing order* sorted\_yi = np.argsort(-train\_class\_distribution.values)**

**for i in sorted\_yi:**

**print('Number of data points in class', i+1, ':',cv\_class\_distribution.values[i], '(', np.round((cv\_class \_distribution.values[i]/y\_cv.shape[0]\*100), 3), '%)')**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 22/159

4/6/2021 temp-161770208870318078 

**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 %)**

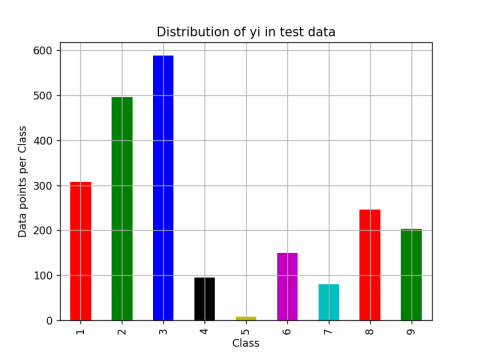
**Number of data points in class 4 : 304 ( 4.371 %)**

**Number of data points in class 7 : 254 ( 3.652 %)**

**Number of data points in class 5 : 27 ( 0.388 %)**

**--------------------------------------------------------------------------------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 23/159

4/6/2021 temp-161770208870318078 

**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 %)**

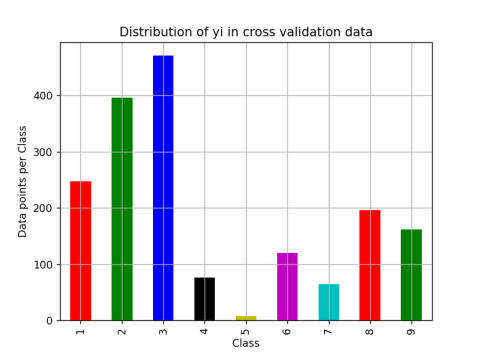
**Number of data points in class 4 : 95 ( 4.37 %)**

**Number of data points in class 7 : 80 ( 3.68 %)**

**Number of data points in class 5 : 8 ( 0.368 %)**

**--------------------------------------------------------------------------------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 24/159

4/6/2021 temp-161770208870318078 

**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 %)**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 25/159

4/6/2021 temp-161770208870318078

**In [ ]: def plot\_confusion\_matrix(test\_y, predict\_y):**

**C = confusion\_matrix(test\_y, predict\_y)**

**print("Number of misclassified points ",(len(test\_y)-np.trace(C))/len(test\_y)\*100)**

***# C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j***

**A =(((C.T)/(C.sum(axis=1))).T)**

***#divid each element of the confusion matrix with the sum of elements in that column***

***# C = [[1, 2],***

***# [3, 4]]***

***# C.T = [[1, 3],***

***# [2, 4]]***

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

***# ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]***

***# [2/3, 4/7]]***

***# ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]***

***# [3/7, 4/7]]***

***# sum of row elements = 1***

**B =(C/C.sum(axis=0))**

***#divid each element of the confusion matrix with the sum of elements in that row***

***# C = [[1, 2],***

***# [3, 4]]***

***# C.sum(axis = 0) axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array # C.sum(axix =0) = [[4, 6]]***

***# (C/C.sum(axis=0)) = [[1/4, 2/6],***

***# [3/4, 4/6]]***

**labels = [1,2,3,4,5,6,7,8,9]**

**cmap=sns.light\_palette("green")**

***# representing A in heatmap format***

**print("-"\*50, "Confusion matrix", "-"\*50)**

**plt.figure(figsize=(10,5))**

**sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels) plt.xlabel('Predicted Class')**

**plt.ylabel('Original Class')**

**plt.show()**

**print("-"\*50, "Precision matrix", "-"\*50)**

**plt.figure(figsize=(10,5))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 26/159

4/6/2021 temp-161770208870318078

**sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels) plt.xlabel('Predicted Class')**

**plt.ylabel('Original Class')**

**plt.show()**

**print("Sum of columns in precision matrix",B.sum(axis=0))**

***# representing B in heatmap format***

**print("-"\*50, "Recall matrix" , "-"\*50)**

**plt.figure(figsize=(10,5))**

**sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels) plt.xlabel('Predicted Class')**

**plt.ylabel('Original Class')**

**plt.show()**

**print("Sum of rows in precision matrix",A.sum(axis=1))**

**4. Machine Learning Models**

**4.1. Machine Leaning Models on bytes files**

**4.1.1. Random Model**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 27/159

4/6/2021 temp-161770208870318078

**In [ ]: *# we need to generate 9 numbers and the sum of numbers should be 1***

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

**test\_data\_len = X\_test.shape[0]**

**cv\_data\_len = X\_cv.shape[0]**

***# we create a output array that has exactly same size as the CV data***

**cv\_predicted\_y = np.zeros((cv\_data\_len,9))**

**for i in range(cv\_data\_len):**

**rand\_probs = np.random.rand(1,9)**

**cv\_predicted\_y[i] = ((rand\_probs/sum(sum(rand\_probs)))[0])**

**print("Log loss on Cross Validation Data using Random Model",log\_loss(y\_cv,cv\_predicted\_y, eps=1e-15))**

***# Test-Set error.***

***#we create a output array that has exactly same as the test data***

**test\_predicted\_y = np.zeros((test\_data\_len,9))**

**for i in range(test\_data\_len):**

**rand\_probs = np.random.rand(1,9)**

**test\_predicted\_y[i] = ((rand\_probs/sum(sum(rand\_probs)))[0])**

**print("Log loss on Test Data using Random Model",log\_loss(y\_test,test\_predicted\_y, eps=1e-15))**

**predicted\_y =np.argmax(test\_predicted\_y, axis=1)**

**plot\_confusion\_matrix(y\_test, predicted\_y+1)**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 28/159

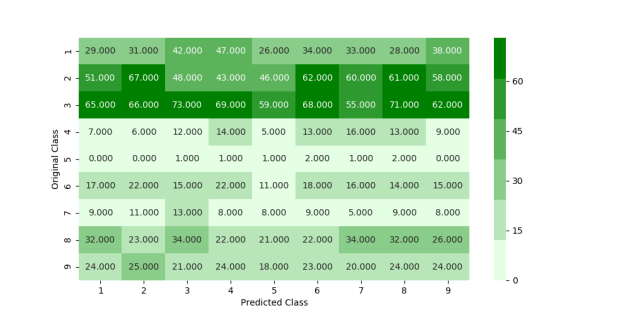
4/6/2021 temp-161770208870318078

**Log loss on Cross Validation Data using Random Model 2.516827727168932**

**Log loss on Test Data using Random Model 2.472894600278911**

**Number of misclassified points 87.90248390064399**

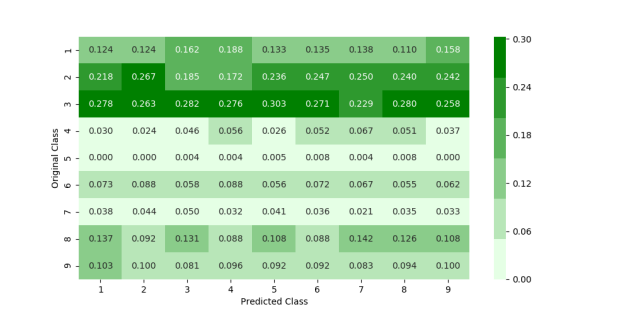
**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

****

**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 29/159

4/6/2021 temp-161770208870318078

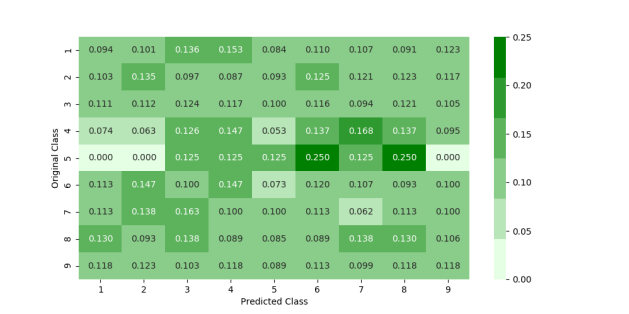


**Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]**

**-------------------------------------------------- Recall matrix -------------------------------------------- ------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 30/159

4/6/2021 temp-161770208870318078



**Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]4.1.2. K Nearest Neighbour Classification**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 31/159

4/6/2021 temp-161770208870318078

**In [ ]: *# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighb ors.KNeighborsClassifier.html***

***# -------------------------***

***# default parameter***

***# KNeighborsClassifier(n\_neighbors=5, weights=’uniform’, algorithm=’auto’, leaf\_size=30, p=2, # metric=’minkowski’, metric\_params=None, n\_jobs=1, \*\*kwargs)***

***# methods of***

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

***# predict(X):Predict the class labels for the provided data***

***# predict\_proba(X):Return probability estimates for the test data X.***

***#-------------------------------------***

***# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geo metric-intuition-with-a-toy-example-1/***

***#-------------------------------------***

***# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.cal ibration.CalibratedClassifierCV.html***

***# ----------------------------***

***# 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))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 32/159

4/6/2021 temp-161770208870318078

**for i in range(len(cv\_log\_error\_array)):**

**print ('log\_loss for k = ',alpha[i],'is',cv\_log\_error\_array[i])**

**best\_alpha = np.argmin(cv\_log\_error\_array)**

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

**ax.plot(alpha, cv\_log\_error\_array,c='g')**

**for i, txt in enumerate(np.round(cv\_log\_error\_array,3)):**

**ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv\_log\_error\_array[i]))**

**plt.grid()**

**plt.title("Cross Validation Error for each alpha")**

**plt.xlabel("Alpha i's")**

**plt.ylabel("Error measure")**

**plt.show()**

**k\_cfl=KNeighborsClassifier(n\_neighbors=alpha[best\_alpha])**

**k\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(k\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print ('For values of best alpha = ', alpha[best\_alpha], "The train log loss is:",log\_loss(y\_train, predict\_y ))**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print('For values of best alpha = ', alpha[best\_alpha], "The cross validation log loss is:",log\_loss(y\_cv, pr edict\_y))**

**predict\_y = sig\_clf.predict\_proba(X\_test)**

**print('For values of best alpha = ', alpha[best\_alpha], "The test log loss is:",log\_loss(y\_test, predict\_y)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 33/159

4/6/2021 temp-161770208870318078

**log\_loss for k = 1 is 0.225386237304**

**log\_loss for k = 3 is 0.230795229168**

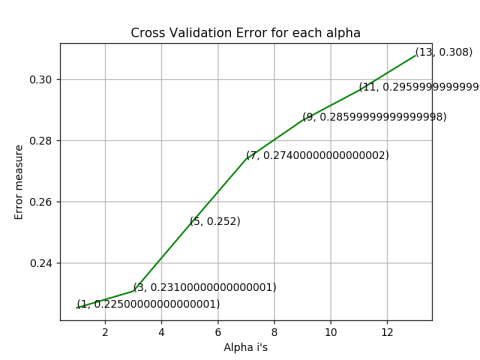
**log\_loss for k = 5 is 0.252421408646**

**log\_loss for k = 7 is 0.273827486888**

**log\_loss for k = 9 is 0.286469181555**

**log\_loss for k = 11 is 0.29623391147**

**log\_loss for k = 13 is 0.307551203154**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 34/159

4/6/2021 temp-161770208870318078

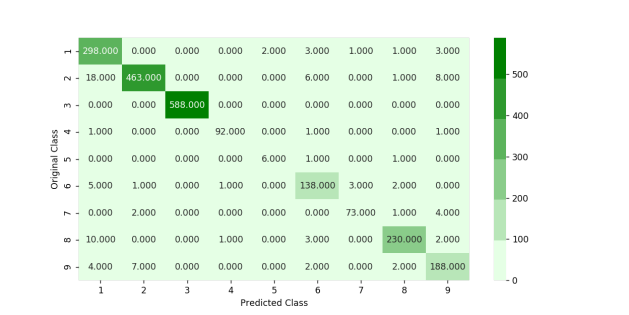
**For values of best alpha = 1 The train log loss is: 0.0782947669247**

**For values of best alpha = 1 The cross validation log loss is: 0.225386237304**

**For values of best alpha = 1 The test log loss is: 0.241508604195**

**Number of misclassified points 4.50781968721**

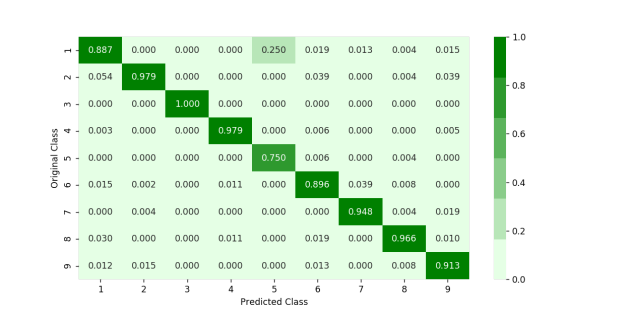
**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

****

**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 35/159

4/6/2021 temp-161770208870318078

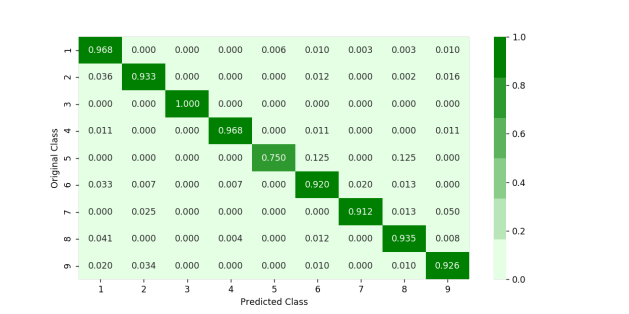


**Sum of columns in precision matrix [ 1. 1. 1. 1. 1. 1. 1. 1. 1.]**

**-------------------------------------------------- Recall matrix -------------------------------------------- ------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 36/159

4/6/2021 temp-161770208870318078



**Sum of rows in precision matrix [ 1. 1. 1. 1. 1. 1. 1. 1. 1.]4.1.3. Logistic Regression**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 37/159

4/6/2021 temp-161770208870318078

**In [ ]: *# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SG DClassifier.html***

***# ------------------------------***

***# default parameters***

***# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1\_ratio=0.15, fit\_intercept=True, max\_iter=None, t ol=None,***

***# shuffle=True, verbose=0, epsilon=0.1, n\_jobs=1, random\_state=None, learning\_rate=’optimal’, eta0=0.0, power \_t=0.5,***

***# class\_weight=None, warm\_start=False, average=False, n\_iter=None)***

***# some of methods***

***# fit(X, y[, coef\_init, intercept\_init, …]) Fit linear model with Stochastic Gradient Descent. # predict(X) Predict class labels for samples in X.***

***#-------------------------------***

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

**alpha = [10 \*\* x for x in range(-5, 10)]**

**cv\_log\_error\_array=[]**

**for i in alpha:**

**logisticR=LogisticRegression(penalty='l2',C=i,class\_weight='balanced')**

**logisticR.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(logisticR, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**cv\_log\_error\_array.append(log\_loss(y\_cv, predict\_y, labels=logisticR.classes\_, eps=1e-15)) print ('log\_loss for c = ',i,'is',log\_loss(y\_cv, predict\_y, labels=logisticR.classes\_, eps=1e-15))**

**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()**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 38/159

4/6/2021 temp-161770208870318078

**logisticR=LogisticRegression(penalty='l2',C=alpha[best\_alpha],class\_weight='balanced') logisticR.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(logisticR, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**pred\_y=sig\_clf.predict(X\_test)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print ('log loss for train data',log\_loss(y\_train, predict\_y, labels=logisticR.classes\_, eps=1e-15)) predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print ('log loss for cv data',log\_loss(y\_cv, predict\_y, labels=logisticR.classes\_, eps=1e-15)) predict\_y = sig\_clf.predict\_proba(X\_test)**

**print ('log loss for test data',log\_loss(y\_test, predict\_y, labels=logisticR.classes\_, eps=1e-15)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 39/159

4/6/2021 temp-161770208870318078

**log\_loss for c = 1e-05 is 1.1477427690814326**

**log\_loss for c = 0.0001 is 1.1477123561014013**

**log\_loss for c = 0.001 is 1.1445515717917563**

**log\_loss for c = 0.01 is 1.118281731721919**

**log\_loss for c = 0.1 is 1.0204633004910921**

**log\_loss for c = 1 is 0.961102329058923**

**log\_loss for c = 10 is 0.9017995673437804**

**log\_loss for c = 100 is 0.8863441394294609**

**log\_loss for c = 1000 is 0.8808767156866277**

**log\_loss for c = 10000 is 0.8958937387197167**

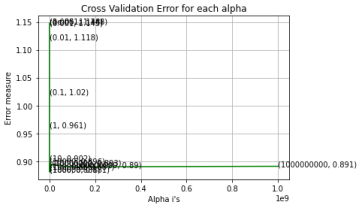
**log\_loss for c = 100000 is 0.8812962451676255**

**log\_loss for c = 1000000 is 0.8865017495697317**

**log\_loss for c = 10000000 is 0.8934364131591847**

**log\_loss for c = 100000000 is 0.8898886342113409**

**log\_loss for c = 1000000000 is 0.8908847757744014**

****

**log loss for train data 0.8736861052915526**

**log loss for cv data 0.8808767156866277**

**log loss for test data 0.8744099779111918**

**Number of misclassified points 29.300827966881325**

**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 40/159

4/6/2021 temp-161770208870318078



**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 41/159

4/6/2021 temp-161770208870318078

**Sum of columns in precision matrix [ 1. 1. 1. 1. nan 1. 1. 1. 1.]**

**-------------------------------------------------- Recall matrix -------------------------------------------- ------**

****

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

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 42/159

4/6/2021 temp-161770208870318078

**In [ ]: *# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SG DClassifier.html***

***# ------------------------------***

***# default parameters***

***# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1\_ratio=0.15, fit\_intercept=True, max\_iter=None, t ol=None,***

***# shuffle=True, verbose=0, epsilon=0.1, n\_jobs=1, random\_state=None, learning\_rate=’optimal’, eta0=0.0, power \_t=0.5,***

***# class\_weight=None, warm\_start=False, average=False, n\_iter=None)***

***# some of methods***

***# fit(X, y[, coef\_init, intercept\_init, …]) Fit linear model with Stochastic Gradient Descent. # predict(X) Predict class labels for samples in X.***

***#-------------------------------***

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

**alpha = [10 \*\* x for x in range(-5, 4)]**

**cv\_log\_error\_array=[]**

**for i in alpha:**

**logisticR=LogisticRegression(penalty='l2',C=i,class\_weight='balanced')**

**logisticR.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(logisticR, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**cv\_log\_error\_array.append(log\_loss(y\_cv, predict\_y, labels=logisticR.classes\_, eps=1e-15))**

**for i in range(len(cv\_log\_error\_array)):**

**print ('log\_loss for c = ',alpha[i],'is',cv\_log\_error\_array[i])**

**best\_alpha = np.argmin(cv\_log\_error\_array)**

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

**ax.plot(alpha, cv\_log\_error\_array,c='g')**

**for i, txt in enumerate(np.round(cv\_log\_error\_array,3)):**

**ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv\_log\_error\_array[i]))**

**plt.grid()**

**plt.title("Cross Validation Error for each alpha")**

**plt.xlabel("Alpha i's")**

**plt.ylabel("Error measure")**

**plt.show()**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 43/159

4/6/2021 temp-161770208870318078

**logisticR=LogisticRegression(penalty='l2',C=alpha[best\_alpha],class\_weight='balanced') logisticR.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(logisticR, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**pred\_y=sig\_clf.predict(X\_test)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print ('log loss for train data',log\_loss(y\_train, predict\_y, labels=logisticR.classes\_, eps=1e-15)) predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print ('log loss for cv data',log\_loss(y\_cv, predict\_y, labels=logisticR.classes\_, eps=1e-15)) predict\_y = sig\_clf.predict\_proba(X\_test)**

**print ('log loss for test data',log\_loss(y\_test, predict\_y, labels=logisticR.classes\_, eps=1e-15)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 44/159

4/6/2021 temp-161770208870318078

**log\_loss for c = 1e-05 is 1.56916911178**

**log\_loss for c = 0.0001 is 1.57336384417**

**log\_loss for c = 0.001 is 1.53598598273**

**log\_loss for c = 0.01 is 1.01720972418**

**log\_loss for c = 0.1 is 0.857766083873**

**log\_loss for c = 1 is 0.711154393309**

**log\_loss for c = 10 is 0.583929522635**

**log\_loss for c = 100 is 0.549929846589**

**log\_loss for c = 1000 is 0.624746769121**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 45/159

4/6/2021 temp-161770208870318078

**log loss for train data 0.498923428696**

**log loss for cv data 0.549929846589**

**log loss for test data 0.528347316704**

**Number of misclassified points 12.3275068997**

**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

****

**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 46/159

4/6/2021 temp-161770208870318078



**Sum of columns in precision matrix [ 1. 1. 1. 1. nan 1. 1. 1. 1.]**

**-------------------------------------------------- Recall matrix -------------------------------------------- ------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 47/159

4/6/2021 temp-161770208870318078



**Sum of rows in precision matrix [ 1. 1. 1. 1. 1. 1. 1. 1. 1.]4.1.4. Random Forest Classifier**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 48/159

4/6/2021 temp-161770208870318078

**In [ ]: *# --------------------------------***

***# default parameters***

***# sklearn.ensemble.RandomForestClassifier(n\_estimators=10, criterion=’gini’, max\_depth=None, min\_samples\_spli t=2,***

***# min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=’auto’, max\_leaf\_nodes=None, min\_impurity\_de crease=0.0,***

***# min\_impurity\_split=None, bootstrap=True, oob\_score=False, n\_jobs=1, random\_state=None, verbose=0, warm\_star t=False,***

***# class\_weight=None)***

***# Some of methods of RandomForestClassifier()***

***# fit(X, y, [sample\_weight]) Fit the SVM model according to the given training data.***

***# predict(X) Perform classification on samples in X.***

***# predict\_proba (X) Perform classification on samples in X.***

***# some of attributes of RandomForestClassifier()***

***# feature\_importances\_ : array of shape = [n\_features]***

***# The feature importances (the higher, the more important the feature).***

***# --------------------------------***

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

***# --------------------------------***

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

**cv\_log\_error\_array=[]**

**train\_log\_error\_array=[]**

**from sklearn.ensemble import RandomForestClassifier**

**for i in alpha:**

**r\_cfl=RandomForestClassifier(n\_estimators=i,random\_state=42,n\_jobs=-1)**

**r\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(r\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**cv\_log\_error\_array.append(log\_loss(y\_cv, predict\_y, labels=r\_cfl.classes\_, eps=1e-15)) print ('log\_loss for c = ',i,'is',log\_loss(y\_cv, predict\_y, labels=logisticR.classes\_, eps=1e-15))**

**best\_alpha = np.argmin(cv\_log\_error\_array)**

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

**ax.plot(alpha, cv\_log\_error\_array,c='g')**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 49/159

4/6/2021 temp-161770208870318078

**for i, txt in enumerate(np.round(cv\_log\_error\_array,3)):**

**ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv\_log\_error\_array[i]))**

**plt.grid()**

**plt.title("Cross Validation Error for each alpha")**

**plt.xlabel("Alpha i's")**

**plt.ylabel("Error measure")**

**plt.show()**

**r\_cfl=RandomForestClassifier(n\_estimators=alpha[best\_alpha],random\_state=42,n\_jobs=-1) r\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(r\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print('For values of best alpha = ', alpha[best\_alpha], "The train log loss is:",log\_loss(y\_train, predict\_y ))**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print('For values of best alpha = ', alpha[best\_alpha], "The cross validation log loss is:",log\_loss(y\_cv, pr edict\_y))**

**predict\_y = sig\_clf.predict\_proba(X\_test)**

**print('For values of best alpha = ', alpha[best\_alpha], "The test log loss is:",log\_loss(y\_test, predict\_y)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 50/159

4/6/2021 temp-161770208870318078

**log\_loss for c = 10 is 0.09218167966430728**

**log\_loss for c = 50 is 0.07606551434778777**

**log\_loss for c = 100 is 0.07767381635567014**

**log\_loss for c = 500 is 0.07593026409685806**

**log\_loss for c = 1000 is 0.07606975076771103**

**log\_loss for c = 2000 is 0.07600040818587882**

**log\_loss for c = 3000 is 0.07589260153327476**

****

**For values of best alpha = 3000 The train log loss is: 0.02581000813607125**

**For values of best alpha = 3000 The cross validation log loss is: 0.07589260153327476 For values of best alpha = 3000 The test log loss is: 0.08256689600125904**

**Number of misclassified points 1.9779208831646733**

**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 51/159

4/6/2021 temp-161770208870318078



**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 52/159

4/6/2021 temp-161770208870318078

**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.]**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 53/159

4/6/2021 temp-161770208870318078

**In [ ]: *# --------------------------------***

***# default parameters***

***# sklearn.ensemble.RandomForestClassifier(n\_estimators=10, criterion=’gini’, max\_depth=None, min\_samples\_spli t=2,***

***# min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=’auto’, max\_leaf\_nodes=None, min\_impurity\_de crease=0.0,***

***# min\_impurity\_split=None, bootstrap=True, oob\_score=False, n\_jobs=1, random\_state=None, verbose=0, warm\_star t=False,***

***# class\_weight=None)***

***# Some of methods of RandomForestClassifier()***

***# fit(X, y, [sample\_weight]) Fit the SVM model according to the given training data.***

***# predict(X) Perform classification on samples in X.***

***# predict\_proba (X) Perform classification on samples in X.***

***# some of attributes of RandomForestClassifier()***

***# feature\_importances\_ : array of shape = [n\_features]***

***# The feature importances (the higher, the more important the feature).***

***# --------------------------------***

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

***# --------------------------------***

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

**cv\_log\_error\_array=[]**

**train\_log\_error\_array=[]**

**from sklearn.ensemble import RandomForestClassifier**

**for i in alpha:**

**r\_cfl=RandomForestClassifier(n\_estimators=i,random\_state=42,n\_jobs=-1)**

**r\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(r\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**cv\_log\_error\_array.append(log\_loss(y\_cv, predict\_y, labels=r\_cfl.classes\_, eps=1e-15))**

**for i in range(len(cv\_log\_error\_array)):**

**print ('log\_loss for c = ',alpha[i],'is',cv\_log\_error\_array[i])**

**best\_alpha = np.argmin(cv\_log\_error\_array)**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 54/159

4/6/2021 temp-161770208870318078

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

**ax.plot(alpha, cv\_log\_error\_array,c='g')**

**for i, txt in enumerate(np.round(cv\_log\_error\_array,3)):**

**ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv\_log\_error\_array[i]))**

**plt.grid()**

**plt.title("Cross Validation Error for each alpha")**

**plt.xlabel("Alpha i's")**

**plt.ylabel("Error measure")**

**plt.show()**

**r\_cfl=RandomForestClassifier(n\_estimators=alpha[best\_alpha],random\_state=42,n\_jobs=-1) r\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(r\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print('For values of best alpha = ', alpha[best\_alpha], "The train log loss is:",log\_loss(y\_train, predict\_y ))**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print('For values of best alpha = ', alpha[best\_alpha], "The cross validation log loss is:",log\_loss(y\_cv, pr edict\_y))**

**predict\_y = sig\_clf.predict\_proba(X\_test)**

**print('For values of best alpha = ', alpha[best\_alpha], "The test log loss is:",log\_loss(y\_test, predict\_y)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 55/159

4/6/2021 temp-161770208870318078

**log\_loss for c = 10 is 0.106357709164**

**log\_loss for c = 50 is 0.0902124124145**

**log\_loss for c = 100 is 0.0895043339776**

**log\_loss for c = 500 is 0.0881420869288**

**log\_loss for c = 1000 is 0.0879849524621**

**log\_loss for c = 2000 is 0.0881566647295**

**log\_loss for c = 3000 is 0.0881318948443**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 56/159

4/6/2021 temp-161770208870318078

**For values of best alpha = 1000 The train log loss is: 0.0266476291801**

**For values of best alpha = 1000 The cross validation log loss is: 0.0879849524621 For values of best alpha = 1000 The test log loss is: 0.0858346961407**

**Number of misclassified points 2.02391904324**

**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

****

**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 57/159

4/6/2021 temp-161770208870318078



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

**-------------------------------------------------- Recall matrix -------------------------------------------- ------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 58/159

4/6/2021 temp-161770208870318078



**Sum of rows in precision matrix [ 1. 1. 1. 1. 1. 1. 1. 1. 1.]4.1.5. XgBoost Classification**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 59/159

4/6/2021 temp-161770208870318078

**In [ ]: *# Training a hyper-parameter tuned Xg-Boost regressor on our train data***

***# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python\_api.html? #xgboost.XGBClassifier***

***# -------------------------***

***# default paramters***

***# class xgboost.XGBClassifier(max\_depth=3, learning\_rate=0.1, n\_estimators=100, silent=True, # objective='binary:logistic', booster='gbtree', n\_jobs=1, nthread=None, gamma=0, min\_child\_weight=1, # max\_delta\_step=0, subsample=1, colsample\_bytree=1, colsample\_bylevel=1, reg\_alpha=0, reg\_lambda=1, # scale\_pos\_weight=1, base\_score=0.5, random\_state=0, seed=None, missing=None, \*\*kwargs)***

***# some of methods of RandomForestRegressor()***

***# fit(X, y, sample\_weight=None, eval\_set=None, eval\_metric=None, early\_stopping\_rounds=None, verbose=True, xg b\_model=None)***

***# get\_params([deep]) Get parameters for this estimator.***

***# predict(data, output\_margin=False, ntree\_limit=0) : Predict with data. NOTE: This function is not thread sa fe.***

***# get\_score(importance\_type='weight') -> get the feature importance***

***# -----------------------***

***# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decis ion-trees-2/***

***# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/ # -----------------------***

**alpha=[10,50,100,500,1000,2000]**

**cv\_log\_error\_array=[]**

**for i in alpha:**

**x\_cfl=XGBClassifier(n\_estimators=i,nthread=-1)**

**x\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(x\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**cv\_log\_error\_array.append(log\_loss(y\_cv, predict\_y, labels=x\_cfl.classes\_, eps=1e-15)) print ('log\_loss for c = ',i,'is',log\_loss(y\_cv, predict\_y, labels=x\_cfl.classes\_, eps=1e-15))**

**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)):**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 60/159

4/6/2021 temp-161770208870318078

**ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv\_log\_error\_array[i]))**

**plt.grid()**

**plt.title("Cross Validation Error for each alpha")**

**plt.xlabel("Alpha i's")**

**plt.ylabel("Error measure")**

**plt.show()**

**x\_cfl=XGBClassifier(n\_estimators=alpha[best\_alpha],nthread=-1)**

**x\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(x\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print ('For values of best alpha = ', alpha[best\_alpha], "The train log loss is:",log\_loss(y\_train, predict\_y ))**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print('For values of best alpha = ', alpha[best\_alpha], "The cross validation log loss is:",log\_loss(y\_cv, pr edict\_y))**

**predict\_y = sig\_clf.predict\_proba(X\_test)**

**print('For values of best alpha = ', alpha[best\_alpha], "The test log loss is:",log\_loss(y\_test, predict\_y)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 61/159

4/6/2021 temp-161770208870318078

**log\_loss for c = 10 is 0.21312094422922967**

**log\_loss for c = 50 is 0.11005569291407905**

**log\_loss for c = 100 is 0.079155858230498**

**log\_loss for c = 500 is 0.0695388029407082**

**log\_loss for c = 1000 is 0.07043320781550406**

**log\_loss for c = 2000 is 0.07082577436190604**

****

**For values of best alpha = 500 The train log loss is: 0.022022904901805872**

**For values of best alpha = 500 The cross validation log loss is: 0.0695388029407082 For values of best alpha = 500 The test log loss is: 0.07797991617448286**

**Number of misclassified points 1.517939282428703**

**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 62/159

4/6/2021 temp-161770208870318078



**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

****

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 63/159

4/6/2021 temp-161770208870318078

**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.]**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 64/159

4/6/2021 temp-161770208870318078

**In [ ]: *# Training a hyper-parameter tuned Xg-Boost regressor on our train data***

***# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python\_api.html? #xgboost.XGBClassifier***

***# -------------------------***

***# default paramters***

***# class xgboost.XGBClassifier(max\_depth=3, learning\_rate=0.1, n\_estimators=100, silent=True, # objective='binary:logistic', booster='gbtree', n\_jobs=1, nthread=None, gamma=0, min\_child\_weight=1, # max\_delta\_step=0, subsample=1, colsample\_bytree=1, colsample\_bylevel=1, reg\_alpha=0, reg\_lambda=1, # scale\_pos\_weight=1, base\_score=0.5, random\_state=0, seed=None, missing=None, \*\*kwargs)***

***# some of methods of RandomForestRegressor()***

***# fit(X, y, sample\_weight=None, eval\_set=None, eval\_metric=None, early\_stopping\_rounds=None, verbose=True, xg b\_model=None)***

***# get\_params([deep]) Get parameters for this estimator.***

***# predict(data, output\_margin=False, ntree\_limit=0) : Predict with data. NOTE: This function is not thread sa fe.***

***# get\_score(importance\_type='weight') -> get the feature importance***

***# -----------------------***

***# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decis ion-trees-2/***

***# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/ # -----------------------***

**alpha=[10,50,100,500,1000,2000]**

**cv\_log\_error\_array=[]**

**for i in alpha:**

**x\_cfl=XGBClassifier(n\_estimators=i,nthread=-1)**

**x\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(x\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**cv\_log\_error\_array.append(log\_loss(y\_cv, predict\_y, labels=x\_cfl.classes\_, eps=1e-15))**

**for i in range(len(cv\_log\_error\_array)):**

**print ('log\_loss for c = ',alpha[i],'is',cv\_log\_error\_array[i])**

**best\_alpha = np.argmin(cv\_log\_error\_array)**

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

**ax.plot(alpha, cv\_log\_error\_array,c='g')**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 65/159

4/6/2021 temp-161770208870318078

**for i, txt in enumerate(np.round(cv\_log\_error\_array,3)):**

**ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv\_log\_error\_array[i]))**

**plt.grid()**

**plt.title("Cross Validation Error for each alpha")**

**plt.xlabel("Alpha i's")**

**plt.ylabel("Error measure")**

**plt.show()**

**x\_cfl=XGBClassifier(n\_estimators=alpha[best\_alpha],nthread=-1)**

**x\_cfl.fit(X\_train,y\_train)**

**sig\_clf = CalibratedClassifierCV(x\_cfl, method="sigmoid")**

**sig\_clf.fit(X\_train, y\_train)**

**predict\_y = sig\_clf.predict\_proba(X\_train)**

**print ('For values of best alpha = ', alpha[best\_alpha], "The train log loss is:",log\_loss(y\_train, predict\_y ))**

**predict\_y = sig\_clf.predict\_proba(X\_cv)**

**print('For values of best alpha = ', alpha[best\_alpha], "The cross validation log loss is:",log\_loss(y\_cv, pr edict\_y))**

**predict\_y = sig\_clf.predict\_proba(X\_test)**

**print('For values of best alpha = ', alpha[best\_alpha], "The test log loss is:",log\_loss(y\_test, predict\_y)) plot\_confusion\_matrix(y\_test, sig\_clf.predict(X\_test))**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 66/159

4/6/2021 temp-161770208870318078

**log\_loss for c = 10 is 0.20615980494**

**log\_loss for c = 50 is 0.123888382365**

**log\_loss for c = 100 is 0.099919437112**

**log\_loss for c = 500 is 0.0931035681289**

**log\_loss for c = 1000 is 0.0933084876012**

**log\_loss for c = 2000 is 0.0938395690309**

****

**For values of best alpha = 500 The train log loss is: 0.0225231805824**

**For values of best alpha = 500 The cross validation log loss is: 0.0931035681289**

**For values of best alpha = 500 The test log loss is: 0.0792067651731**

**Number of misclassified points 1.24195032199**

**-------------------------------------------------- Confusion matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 67/159

4/6/2021 temp-161770208870318078



**-------------------------------------------------- Precision matrix ----------------------------------------- ---------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 68/159

4/6/2021 temp-161770208870318078



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

**-------------------------------------------------- Recall matrix -------------------------------------------- ------**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 69/159

4/6/2021 temp-161770208870318078



**Sum of rows in precision matrix [ 1. 1. 1. 1. 1. 1. 1. 1. 1.]4.1.5. XgBoost Classification with best hyper parameters using RandomSearch**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 70/159

4/6/2021 temp-161770208870318078

**In [ ]: *# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/* x\_cfl=XGBClassifier()**

**prams={**

**'learning\_rate':[0.01,0.03,0.05,0.1,0.15,0.2],**

**'n\_estimators':[100,200,500,1000,2000],**

**'max\_depth':[3,5,10],**

**'colsample\_bytree':[0.1,0.3,0.5,1],**

**'subsample':[0.1,0.3,0.5,1]**

**}**

**random\_cfl1=RandomizedSearchCV(x\_cfl,param\_distributions=prams,verbose=10,n\_jobs=-1,)**

**random\_cfl1.fit(X\_train,y\_train)**

**Fitting 3 folds for each of 10 candidates, totalling 30 fits**

**[Parallel(n\_jobs=-1)]: Done 2 tasks | elapsed: 26.5s**

**[Parallel(n\_jobs=-1)]: Done 9 tasks | elapsed: 5.8min**

**[Parallel(n\_jobs=-1)]: Done 19 out of 30 | elapsed: 9.3min remaining: 5.4min**

**[Parallel(n\_jobs=-1)]: Done 23 out of 30 | elapsed: 10.1min remaining: 3.1min**

**[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: 14.2min finished**

**Out[ ]: RandomizedSearchCV(cv=None, error\_score='raise',**

**estimator=XGBClassifier(base\_score=0.5, colsample\_bylevel=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, nthread=-1,**

**objective='binary:logistic', reg\_alpha=0, reg\_lambda=1,**

**scale\_pos\_weight=1, seed=0, silent=True, subsample=1),**

**fit\_params=None, iid=True, n\_iter=10, n\_jobs=-1,**

**param\_distributions={'learning\_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n\_estimators': [100, 20 0, 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]},**

**pre\_dispatch='2\*n\_jobs', random\_state=None, refit=True,**

**return\_train\_score=True, scoring=None, verbose=10)**

**In [ ]: print (random\_cfl1.best\_params\_)**

**{'subsample': 1, 'n\_estimators': 500, 'max\_depth': 5, 'learning\_rate': 0.05, 'colsample\_bytree': 0.5}**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 71/159

4/6/2021 temp-161770208870318078

**In [ ]: *# Training a hyper-parameter tuned Xg-Boost regressor on our train data***

***# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python\_api.html? #xgboost.XGBClassifier***

***# -------------------------***

***# default paramters***

***# class xgboost.XGBClassifier(max\_depth=3, learning\_rate=0.1, n\_estimators=100, silent=True, # objective='binary:logistic', booster='gbtree', n\_jobs=1, nthread=None, gamma=0, min\_child\_weight=1, # max\_delta\_step=0, subsample=1, colsample\_bytree=1, colsample\_bylevel=1, reg\_alpha=0, reg\_lambda=1, # scale\_pos\_weight=1, base\_score=0.5, random\_state=0, seed=None, missing=None, \*\*kwargs)***

***# some of methods of RandomForestRegressor()***

***# fit(X, y, sample\_weight=None, eval\_set=None, eval\_metric=None, early\_stopping\_rounds=None, verbose=True, xg b\_model=None)***

***# get\_params([deep]) Get parameters for this estimator.***

***# predict(data, output\_margin=False, ntree\_limit=0) : Predict with data. NOTE: This function is not thread sa fe.***

***# get\_score(importance\_type='weight') -> get the feature importance***

***# -----------------------***

***# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/ # -----------------------***

**x\_cfl=XGBClassifier(n\_estimators=2000, learning\_rate=0.05, colsample\_bytree=1, max\_depth=3)**

**x\_cfl.fit(X\_train,y\_train)**

**c\_cfl=CalibratedClassifierCV(x\_cfl,method='sigmoid')**

**c\_cfl.fit(X\_train,y\_train)**

**predict\_y = c\_cfl.predict\_proba(X\_train)**

**print ('train loss',log\_loss(y\_train, predict\_y))**

**predict\_y = c\_cfl.predict\_proba(X\_cv)**

**print ('cv loss',log\_loss(y\_cv, predict\_y))**

**predict\_y = c\_cfl.predict\_proba(X\_test)**

**print ('test loss',log\_loss(y\_test, predict\_y))**

**train loss 0.022540976086**

**cv loss 0.0928710624158**

**test loss 0.0782688587098**

**4.2 Modeling with .asm files**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 72/159

4/6/2021 temp-161770208870318078

**There are 10868 files of asm**

**All the files make up about 150 GB**

**The asm files contains :**

**1. Address**

**2. Segments**

**3. Opcodes**

**4. Registers**

**5. function calls**

**6. APIs**

**With the help of parallel processing we extracted all the features.In parallel we can use all the cores that are p resent in our computer.**

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

**We read the top solutions and handpicked the features from those papers/videos/blogs.**

**Refer:https://www.kaggle.com/c/malware-classification/discussion**

**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**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 73/159

4/6/2021 temp-161770208870318078

**In [ ]: *#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')**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 74/159

4/6/2021 temp-161770208870318078

**In [ ]: *#http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html***

**def firstprocess():**

***#The prefixes tells about the segments that are present in the asm files***

***#There are 450 segments(approx) present in all asm files.***

***#this prefixes are best segments that gives us best values.***

***#https://en.wikipedia.org/wiki/Data\_segment***

**prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','. reloc:','.BSS:','.CODE']**

***#this are opcodes that are used to get best results***

***#https://en.wikipedia.org/wiki/X86\_instruction\_listings***

**opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']**

***#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]:**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 75/159

4/6/2021 temp-161770208870318078

**prefixescount[i]+=1**

**line=line[1:]**

***#counting the opcodes in each and every line***

**for i in range(len(opcodes)):**

**if any(opcodes[i]==li for li in line):**

**features.append(opcodes[i])**

**opcodescount[i]+=1**

***#counting registers in the line***

**for i in range(len(registers)):**

**for li in line:**

***# we will use registers only in 'text' and 'CODE' segments***

**if registers[i] in li and ('text' in l or 'CODE' in l):**

**registerscount[i]+=1**

***#counting keywords in the line***

**for i in range(len(keywords)):**

**for li in line:**

**if keywords[i] in li:**

**keywordcount[i]+=1**

***#pushing the values into the file after reading whole file***

**for prefix in prefixescount:**

**file1.write(str(prefix)+",")**

**for opcode in opcodescount:**

**file1.write(str(opcode)+",")**

**for register in registerscount:**

**file1.write(str(register)+",")**

**for key in keywordcount:**

**file1.write(str(key)+",")**

**file1.write("\n")**

**file1.close()**

***#same as above***

**def secondprocess():**

**prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','. reloc:','.BSS:','.CODE']**

**opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx'] 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)**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 76/159

4/6/2021 temp-161770208870318078

**opcodescount=np.zeros(len(opcodes),dtype=int)**

**keywordcount=np.zeros(len(keywords),dtype=int)**

**registerscount=np.zeros(len(registers),dtype=int)**

**features=[]**

**f2=f.split('.')[0]**

**file1.write(f2+",")**

**opcodefile.write(f2+" ")**

**with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:**

**for lines in fli:**

**line=lines.rstrip().split()**

**l=line[0]**

**for i in range(len(prefixes)):**

**if prefixes[i] in line[0]:**

**prefixescount[i]+=1**

**line=line[1:]**

**for i in range(len(opcodes)):**

**if any(opcodes[i]==li for li in line):**

**features.append(opcodes[i])**

**opcodescount[i]+=1**

**for i in range(len(registers)):**

**for li in line:**

**if registers[i] in li and ('text' in l or 'CODE' in l):**

**registerscount[i]+=1**

**for i in range(len(keywords)):**

**for li in line:**

**if keywords[i] in li:**

**keywordcount[i]+=1**

**for prefix in prefixescount:**

**file1.write(str(prefix)+",")**

**for opcode in opcodescount:**

**file1.write(str(opcode)+",")**

**for register in registerscount:**

**file1.write(str(register)+",")**

**for key in keywordcount:**

**file1.write(str(key)+",")**

**file1.write("\n")**

**file1.close()**

***# same as smallprocess() functions***

**def thirdprocess():**

**prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','. reloc:','.BSS:','.CODE']**

**opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul',**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 77/159

4/6/2021 temp-161770208870318078

**'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx'] keywords = ['.dll','std::',':dword']**

**registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip'] 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")**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 78/159

4/6/2021 temp-161770208870318078

**file1.close()**

**def fourthprocess():**

**prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','. reloc:','.BSS:','.CODE']**

**opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx'] 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)+",")**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 79/159

4/6/2021 temp-161770208870318078

**for opcode in opcodescount:**

**file1.write(str(opcode)+",")**

**for register in registerscount:**

**file1.write(str(register)+",")**

**for key in keywordcount:**

**file1.write(str(key)+",")**

**file1.write("\n")**

**file1.close()**

**def fifthprocess():**

**prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','. reloc:','.BSS:','.CODE']**

**opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx'] 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):**

https://htmtopdf.herokuapp.com/ipynbviewer/temp/6e92ad7338db9b0d5d687fda377b9086/MicrosoftMalwareDetection.html?t=1617702142216 80/159