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**

1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

2.1.2. Example Data Point

** .asm file **

```
.text:00401000                                  assume es:nothing, ss:nothing, ds:_data,
s:nothing, gs:nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24 08                          lea     eax, [esp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                                mov     esi, ecx
.text:00401008 E8 1C 1B 00 00                           call    ??
0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08 BB 42 00                        mov     dword ptr [esi], offset o
f_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 o
f_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 o
f_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    ; operato
delete(void *)
.text:0040104B 83 C4 04                             add     esp, 4
.text:0040104E
.text:0040104E                              loc_40104E:                 ; CODE XREF:
.text:00401043 j
.text:0040104E 8B C6                                mov     eax, esi
.text:00401050 5E                               pop     esi
.text:00401051 C2 04 00                             retn    4
.text:00401051                              ; ---------------------------------------------
-------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
```

```
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 20 00 02 00 04
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
```

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

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

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf

https://www.dropbox.com/sh/gfqzv0ckgs4i1bf/AABbEeihEjvvuQg2nu_piB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [0]:

```
!pip install notebook --upgrade --user
```

```
Collecting notebook
  Downloading
https://files.pythonhosted.org/packages/f5/69/d2ffaf7efc20ce47469187e3a41e6e03e17b45de5a6559f4e7ab3
5e1/notebook-6.0.2-py3-none-any.whl (9.7MB)
    100% |████████████████████████████████| 9.7MB 4.2MB/s eta 0:00:01
Requirement already satisfied, skipping upgrade: pyzmq>=17 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (17.1.2)
Requirement already satisfied, skipping upgrade: traitlets>=4.2.1 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (4.3.2)
Requirement already satisfied, skipping upgrade: terminado>=0.8.1 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (0.8.1)
Collecting jupyter-core>=4.6.0 (from notebook)
  Downloading
https://files.pythonhosted.org/packages/fb/82/86437f661875e30682e99d04c13ba6c216f86f5f6ca6ef212d3ee
a11/jupyter_core-4.6.1-py2.py3-none-any.whl (82kB)
    100% |████████████████████████████████| 92kB 38.3MB/s ta 0:00:01
Requirement already satisfied, skipping upgrade: prometheus-client in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (0.5.0)
Requirement already satisfied, skipping upgrade: nbconvert in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (5.3.1)
Collecting jupyter-client>=5.3.4 (from notebook)
  Downloading
https://files.pythonhosted.org/packages/13/81/fe0eee1bcf949851a120254b1f530ae1e01bdde2d3ab9710c6ff8
061/jupyter_client-5.3.4-py2.py3-none-any.whl (92kB)
    100% |████████████████████████████████| 92kB 37.1MB/s ta 0:00:01
Requirement already satisfied, skipping upgrade: nbformat in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (4.4.0)
Requirement already satisfied, skipping upgrade: ipykernel in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (5.1.0)
Requirement already satisfied, skipping upgrade: ipython-genutils in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (0.2.0)
Requirement already satisfied, skipping upgrade: Send2Trash in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (1.5.0)
Requirement already satisfied, skipping upgrade: tornado>=5.0 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (5.1.1)
Requirement already satisfied, skipping upgrade: jinja2 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from notebook) (2.10)
Requirement already satisfied, skipping upgrade: six in /opt/conda/envs/fastai/lib/python3.6/site-
packages (from traitlets>=4.2.1->notebook) (1.12.0)
Requirement already satisfied, skipping upgrade: decorator in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from traitlets>=4.2.1->notebook) (4.3.0)
Requirement already satisfied, skipping upgrade: bleach in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbconvert->notebook) (3.1.0)
Requirement already satisfied, skipping upgrade: pygments in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbconvert->notebook) (2.3.1)
Requirement already satisfied, skipping upgrade: entrypoints>=0.2.2 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbconvert->notebook) (0.3)
Requirement already satisfied, skipping upgrade: testpath in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbconvert->notebook) (0.4.2)
Requirement already satisfied, skipping upgrade: pandocfilters>=1.4.1 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbconvert->notebook) (1.4.2)
Requirement already satisfied, skipping upgrade: mistune>=0.7.4 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbconvert->notebook) (0.8.4)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from jupyter-client>=5.3.4->notebook) (2.7.5)
Requirement already satisfied, skipping upgrade: jsonschema!=2.5.0,>=2.4 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from nbformat->notebook) (3.0.0a3)
Requirement already satisfied, skipping upgrade: ipython>=5.0.0 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipykernel->notebook) (7.2.0)
Requirement already satisfied, skipping upgrade: MarkupSafe>=0.23 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from jinja2->notebook) (1.1.0)
Requirement already satisfied, skipping upgrade: webencodings in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from bleach->nbconvert->notebook) (0.5.1)
Requirement already satisfied, skipping upgrade: attrs>=17.4.0 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat-
>notebook) (18.2.0)
```

```
>notebook) (18.2.0)
Requirement already satisfied, skipping upgrade: pyrsistent>=0.14.0 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat-
>notebook) (0.14.9)
Requirement already satisfied, skipping upgrade: setuptools>=18.5 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipython>=5.0.0->ipykernel->notebook)
(40.6.3)
Requirement already satisfied, skipping upgrade: jedi>=0.10 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipython>=5.0.0->ipykernel->notebook)
(0.13.2)
Requirement already satisfied, skipping upgrade: pickleshare in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipython>=5.0.0->ipykernel->notebook)
(0.7.5)
Requirement already satisfied, skipping upgrade: prompt_toolkit<2.1.0,>=2.0.0 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipython>=5.0.0->ipykernel->notebook)
(2.0.7)
Requirement already satisfied, skipping upgrade: backcall in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipython>=5.0.0->ipykernel->notebook)
(0.1.0)
Requirement already satisfied, skipping upgrade: pexpect in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from ipython>=5.0.0->ipykernel->notebook)
(4.6.0)
Requirement already satisfied, skipping upgrade: parso>=0.3.0 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from jedi>=0.10->ipython>=5.0.0->ipykernel-
>notebook) (0.3.1)
Requirement already satisfied, skipping upgrade: wcwidth in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from prompt_toolkit<2.1.0,>=2.0.0-
>ipython>=5.0.0->ipykernel->notebook) (0.1.7)
Requirement already satisfied, skipping upgrade: ptyprocess>=0.5 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from pexpect->ipython>=5.0.0->ipykernel-
>notebook) (0.6.0)
Installing collected packages: jupyter-core, jupyter-client, notebook
  The scripts jupyter, jupyter-migrate and jupyter-troubleshoot are installed in
'/root/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn
-script-location.
  The scripts jupyter-kernel, jupyter-kernelspec and jupyter-run are installed in
'/root/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn
-script-location.
  The scripts jupyter-bundlerextension, jupyter-nbextension, jupyter-notebook and jupyter-serverex
tension are installed in '/root/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn
-script-location.
Successfully installed jupyter-client-5.3.4 jupyter-core-4.6.1 notebook-6.0.2
```

In [0]:

```
!pip install --upgrade pandas --user
```

```
Requirement already up-to-date: pandas in /root/.local/lib/python3.6/site-packages (0.25.3)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from pandas) (2018.9)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from pandas) (2.7.5)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from pandas) (1.15.4)
Requirement already satisfied, skipping upgrade: six>=1.5 in
/opt/conda/envs/fastai/lib/python3.6/site-packages (from python-dateutil>=2.6.1->pandas) (1.12.0)
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
```

```
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from nltk.util import ngrams
from tqdm import tqdm
from dask import dataframe as dd
import pickle
```

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
#install credentials
#!pip install kaggle
```

In [0]:

```
#upload the creadentials
from google.colab import files
files.upload()
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[0]:

```
{'kaggle.json': b'{"username":"psksaiteja","key":"a41c3a50e829338ef9e68621bdc89d4c"}'}
```

In [0]:

```
#before importing the dataset we want to use this code
#the kaggle api client expects this file to be in ~/.kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/kaggle.json
#this permission change avoid a warning on kaggle tool startup
!chmod 600 ~/.kaggle/kaggle.json
```

In [0]:

```
#import the dataset we want to use for our project
!kaggle competitions download -c malware-classification
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.5.6 /
client 1.5.4)
Downloading sampleSubmission.csv to /content
  0% 0.00/2.01M [00:00<?, ?B/s]
100% 2.01M/2.01M [00:00<00:00, 66.4MB/s]
Downloading trainLabels.csv to /content
  0% 0.00/265k [00:00<?, ?B/s]
100% 265k/265k [00:00<00:00, 61.7MB/s]
Downloading train.7z to /content
100% 17.5G/17.5G [08:33<00:00, 24.4MB/s]
100% 17.5G/17.5G [08:33<00:00, 36.6MB/s]
Downloading test.7z to /content
100% 17.8G/17.8G [03:41<00:00, 35.2MB/s]
100% 17.8G/17.8G [03:42<00:00, 85.8MB/s]
Downloading dataSample.7z to /content
  0% 0.00/4.06M [00:00<?, ?B/s]
100% 4.06M/4.06M [00:00<00:00, 66.9MB/s]
```

In [0]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 6.3; W
in64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" --header="Ac
cept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3" --header="Accept-Language: en-GB,en-US;q=0.9,en;q=0.8" --header="Referer:
https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-competitions-
data/kaggle/4117/train.7z?GoogleAccessId=web-data@kaggle-
161607.iam.gserviceaccount.com&Expires=1573661427&Signature=KN5V9Hvurhosvl%2B0lspgzqQSaGECU1uWx4RvF
wcpoQmtyiebmvOHetldvj17eemgxAaDJU5RpX4p3ZsnPmEZBXe%2BheRDyIScfvBKLyNU9xiTtW0y6e%2FRJelqurYo8i8QQpsA
D54M0aAY%2FuIYSb8kTN5oQhSU%2BUaQi6aVWbtf3hOde7oe6WuaJiiSutAg%2BmvFeVccpLXMbApJRtj60LNahhP%2F4e38%2E
oQeBwLupTh3TZKe3b3gpYgzAv0i5%2BgoDFGSap0tdysAJ37%2FNcdIL9gq813ZT35I5OHW9xFTI5X9fYsnJJKlLryluAlF95uI
AOUf2%2BaMqkzA%3D%3D" -O "train.7z" -c
```

```
--2019-11-10 16:11:15--  https://storage.googleapis.com/kaggle-competitions-
data/kaggle/4117/train.7z?GoogleAccessId=web-data@kaggle-
161607.iam.gserviceaccount.com&Expires=1573661427&Signature=KN5V9Hvurhosvl%2B0lspgzqQSaGECU1uWx4RvF
wcpoQmtyiebmvOHetldvj17eemgxAaDJU5RpX4p3ZsnPmEZBXe%2BheRDyIScfvBKLyNU9xiTtW0y6e%2FRJelqurYo8i8QQpsA
D54M0aAY%2FuIYSb8kTN5oQhSU%2BUaQi6aVWbtf3hOde7oe6WuaJiiSutAg%2BmvFeVccpLXMbApJRtj60LNahhP%2F4e38%2E
oQeBwLupTh3TZKe3b3gpYgzAv0i5%2BgoDFGSap0tdysAJ37%2FNcdIL9gq813ZT35I5OHW9xFTI5X9fYsnJJKlLryluAlF95uI
AOUf2%2BaMqkzA%3D%3D
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.142.128,
2607:f8b0:400e:c08::80
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.142.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18810691091 (18G) [application/x-7z-compressed]
Saving to: 'train.7z'

train.7z            100%[===================>]  17.52G  64.5MB/s    in 6m 26s

2019-11-10 16:17:41 (46.5 MB/s) - 'train.7z' saved [18810691091/18810691091]
```

In [0]:

```
!ls
```

```
asmoutputfile.csv        result_with_size.csv  trainLabels.csv
MicrosoftMalwareDetection_FFF2.ipynb  train.7z      tutorials
```

In [0]:

```
# Unzip the 7zip files
# -d: which file to un7zip
#!p7zip -d train.7z
```

In [0]:

```
#if any face any incompletee plots  in confusion matrix install this version of matplotlib
#!pip install matplotlib==3.1.0
```

In [0]:

```
#separating byte files and asm files

source = 'train'
destination_b = 'byteFiles'
destination_a = '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_b):
    os.makedirs(destination_b)
if not os.path.isdir(destination_a):
    os.makedirs(destination_a)

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

In [0]:

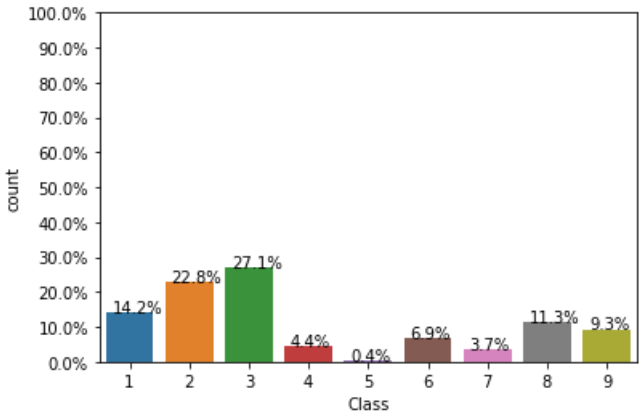
```
# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    data_files = os.listdir(source)
    for file in data_files:
        if (file.endswith("bytes")):
            shutil.move(source+'//'+file,destination_b)
        if (file.endswith("asm")):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

```
Files Moved........
```

3.1. Distribution of malware classes in whole data set

In [0]:

```
%matplotlib inline
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()
```



In [0]:

```
Y.head()
```

Out[0]:

	Id	Class
0	01kcPWA9K2BOxQeS5Rju	1
1	04EjIdbPV5e1XroFOpiN	1
2	05EeG39MTRrl6VY21DPd	1
3	05rJTUWYAKNegBk2wE8X	1
4	0AnoOZDNbPXlr2MRBSCJ	1

3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [0]:

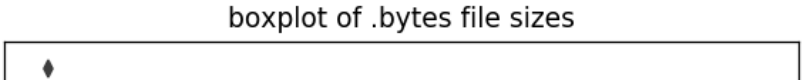
```
#file sizes of byte files

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())
```

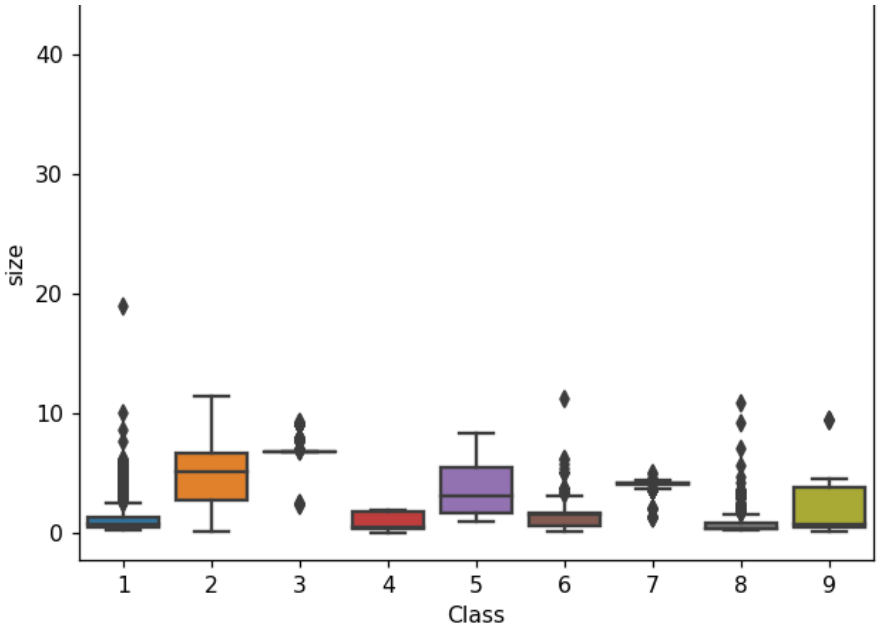
```
                     ID      size  Class
0  01azqd4InC7m9JpocGv5  4.234863      9
1  01IsoiSMh5gxyDYTl4CB  5.538818      2
2  01jsnpXSAlgw6aPeDxrU  3.887939      9
3  01kcPWA9K2BOxQeS5Rju  0.574219      1
4  01SuzwMJEIXsK7A8dQbl  0.370850      8
```

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

In [0]:

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





3.2.3 feature extraction from byte files

In [0]:

```
#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,
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,
e,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,5
,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
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,
5,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,c
,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
ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")
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 [0]:

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

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8	f9	fa	fb	fc
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687	3101	3211	3097	2758
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536	439	281	302	7639

2 rows × 258 columns

In [0]:

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

Out[0]:

	ID	size	Class
0	01azqd4InC7m9JpocGv5	4.234863	9
1	01IsoiSMh5gxyDYTI4CB	5.538818	2

In [0]:

```
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)
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	17001

2 rows × 260 columns

In [0]:

```
byte_features_with_size = pd.read_csv('result_with_size.csv',index_col=0)
byte_features_with_size.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	
0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	1700
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2242	2885	2863	2471	2786	2680
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	485	462	516	1133	471	761
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	350	209	239	653	221	242

5 rows × 260 columns

In [0]:

```
# 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)
```

In [0]:

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

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	
0	01azqd4lnC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.

2 rows × 260 columns

In [0]:

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

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...
0	01azqd4lnC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...
1	01IsoiSMh5gxyDYTI4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...

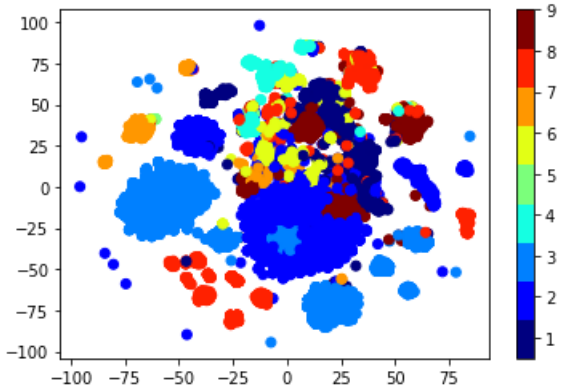
5 rows × 260 columns

3.2.4 Multivariate Analysis

In [0]:

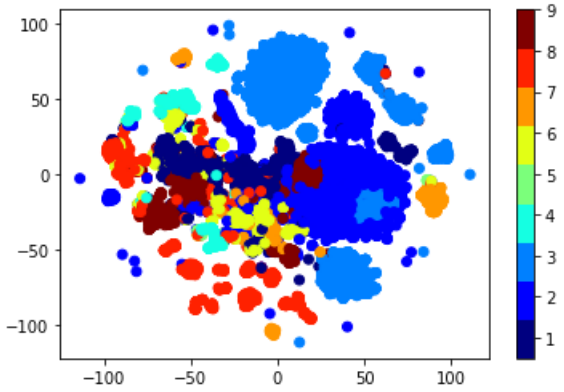
```
#multivariate analysis on byte files
#this is with perplexity 50
```

```
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()
```



In [0]:

```
#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

In [0]:

```
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,str
atify=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)
```

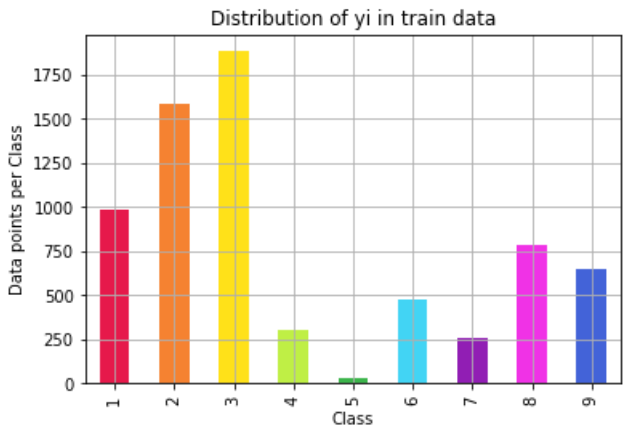
In [0]:

```
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])
```

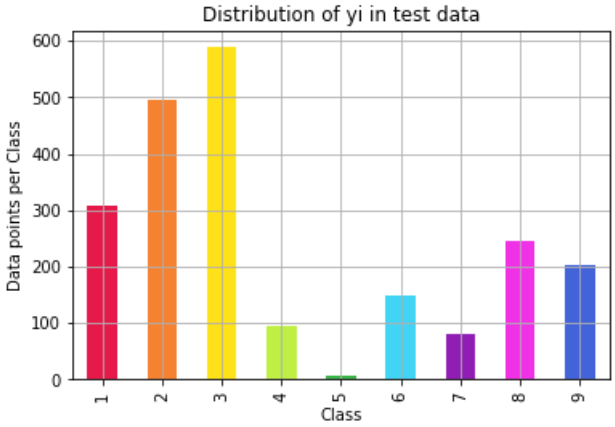
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [0]:

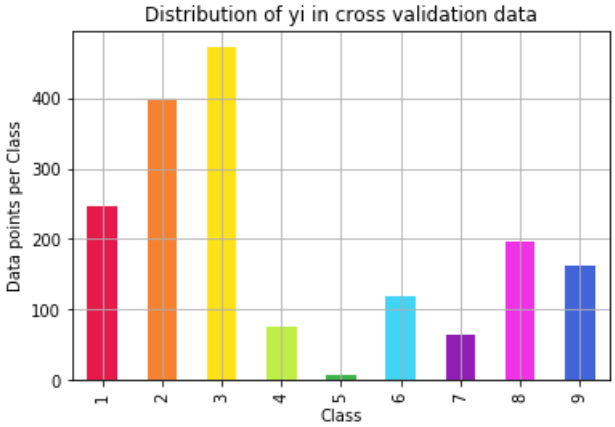
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = y_train.value_counts().sort_index()
test_class_distribution = y_test.value_counts().sort_index()
cv_class_distribution = y_cv.value_counts().sort_index()
my_colors = ['#e6194B','#f58231','#ffe119','#bfef45','#3cb44b','#42d4f4','#911eb4','#f032e6','#436
3d8']

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.ro
und((train_class_distribution.values[i]/y_train.shape[0]*100), 3), '%)')


print('-'*80)
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.rou
nd((test_class_distribution.values[i]/y_test.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round
((cv_class_distribution.values[i]/y_cv.shape[0]*100), 3), '%)')
```



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




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




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

In [0]:

```
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=(15,5))
    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=(15,5))
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of columns in precision matrix",B.sum(axis=0))

    # representing B in heatmap format
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(15,5))
    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

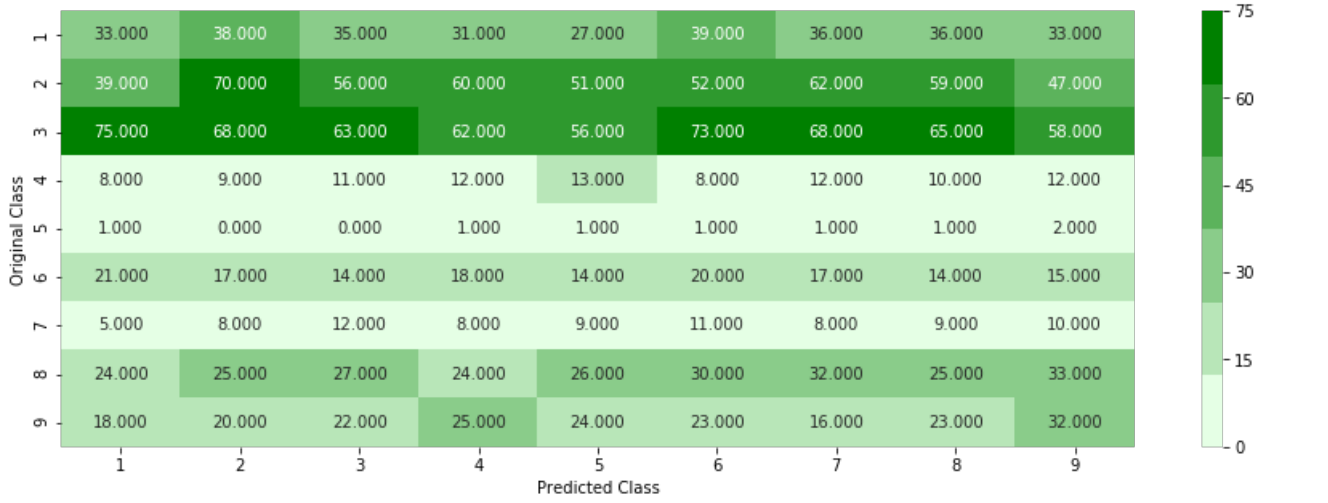
4.1.1. Random Model

In [0]:

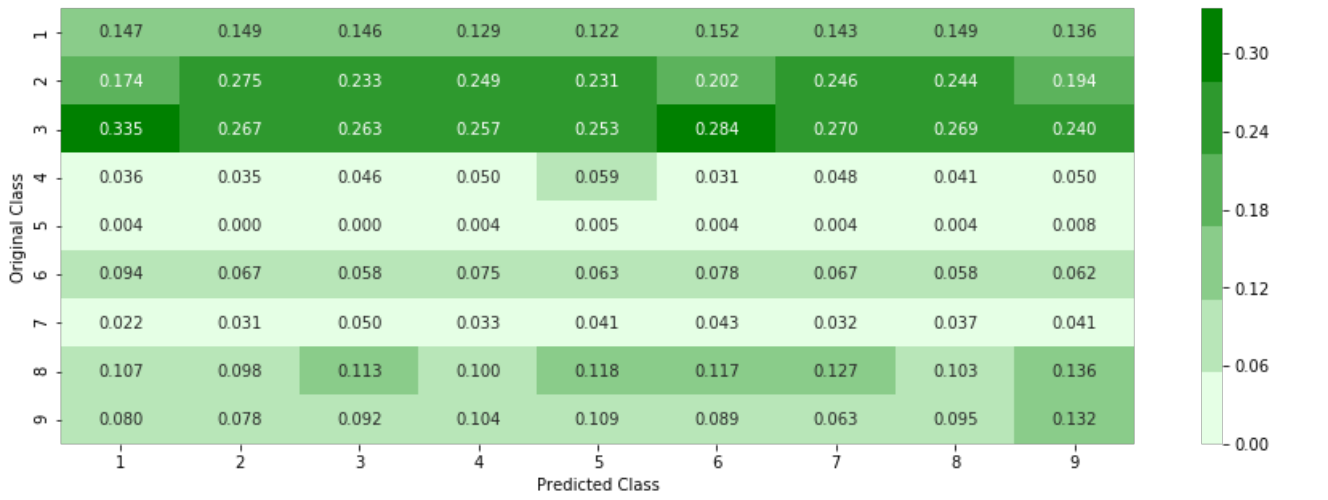
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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)
```

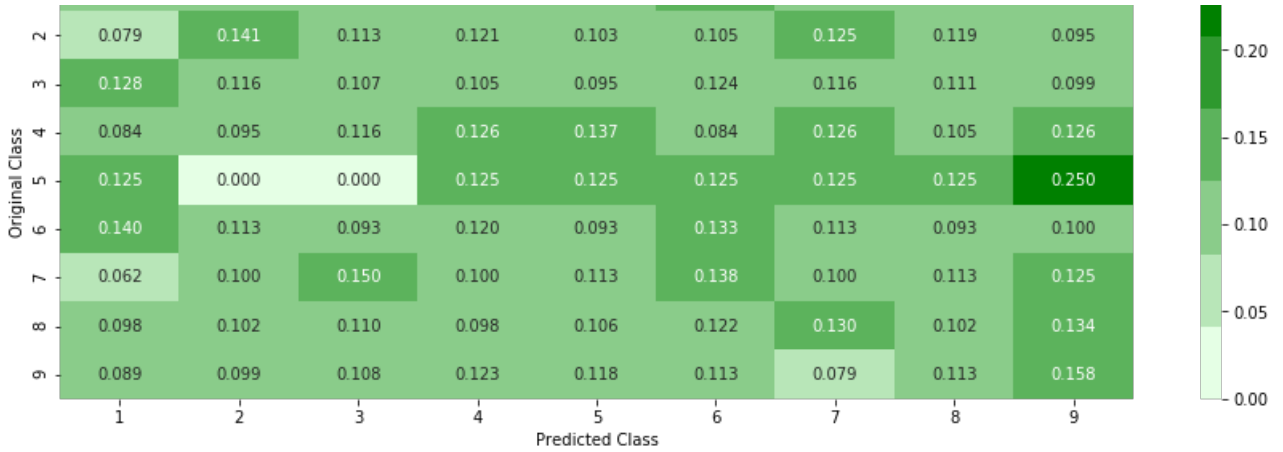
```
Log loss on Cross Validation Data using Random Model 2.4701231563815647
Log loss on Test Data using Random Model 2.4623594728613387
Number of misclassified points  87.85648574057038
------------------------------------------------- Confusion matrix -----------------------------------
----------------
```




```
------------------------------------------------- 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.]

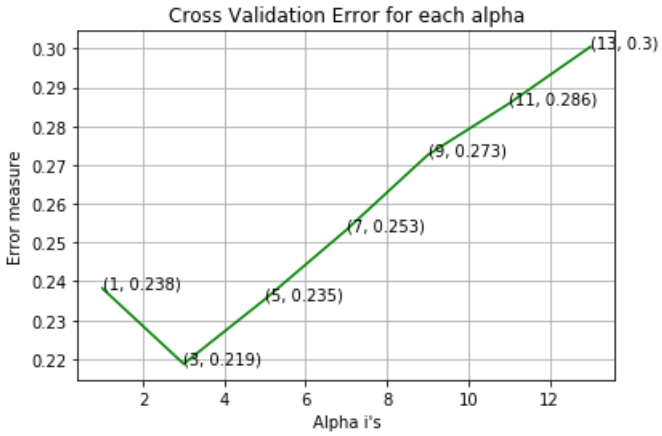
4.1.2. K Nearest Neighbour Classification

In [0]:

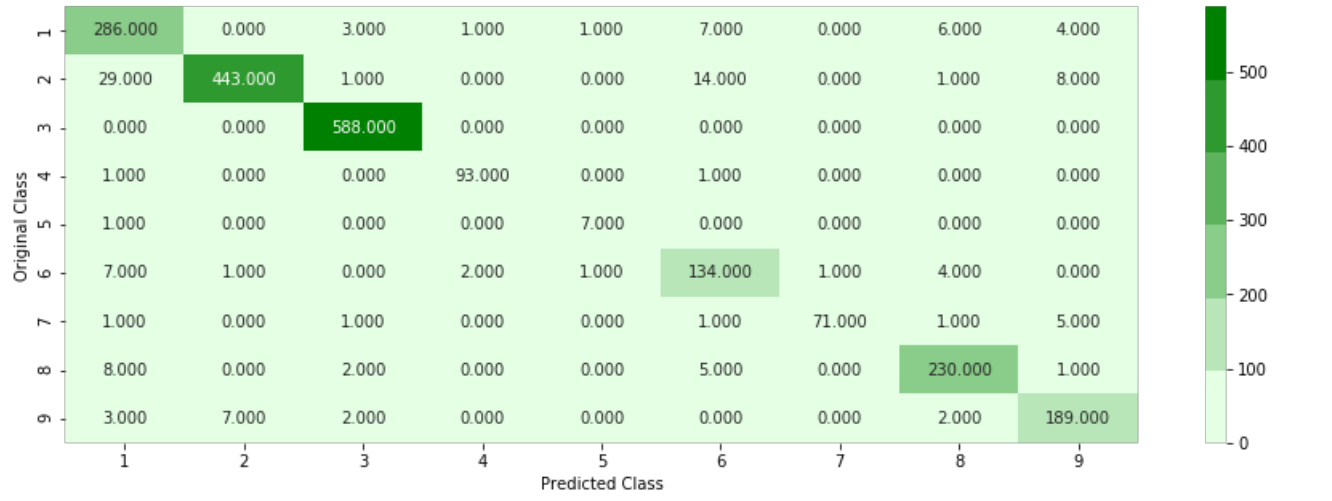
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', 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-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.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))

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_lo
ss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

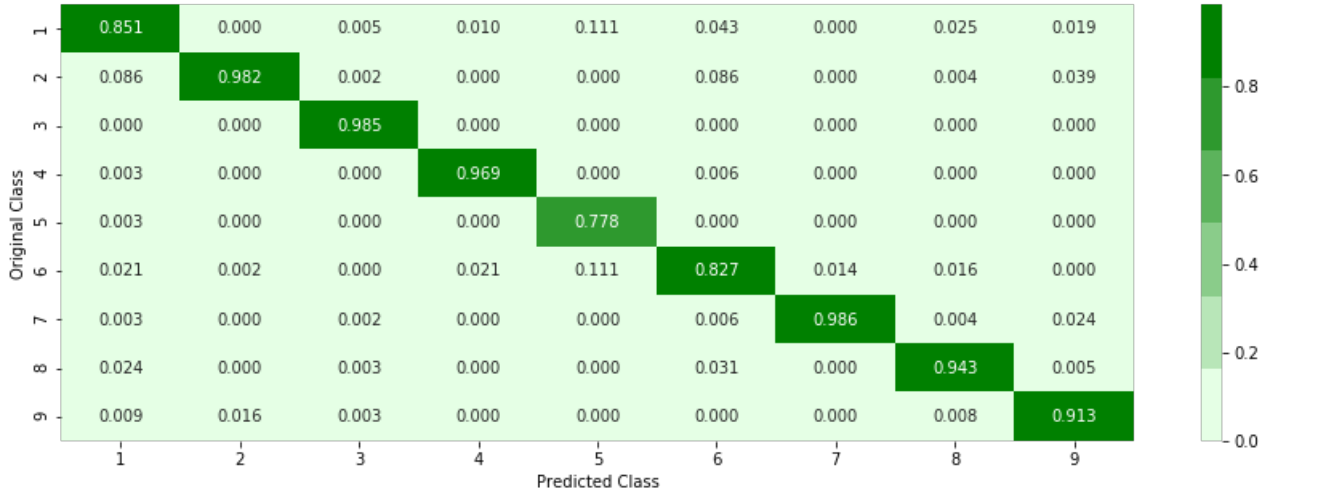
```
log_loss for k =  1 is 0.23821368197590012
log_loss for k =  3 is 0.2187440766511307
log_loss for k =  5 is 0.2354082237963733
log_loss for k =  7 is 0.25326386327959516
log_loss for k =  9 is 0.27256694689600247
log_loss for k =  11 is 0.28583665874948105
log_loss for k =  13 is 0.30045940255483244
```



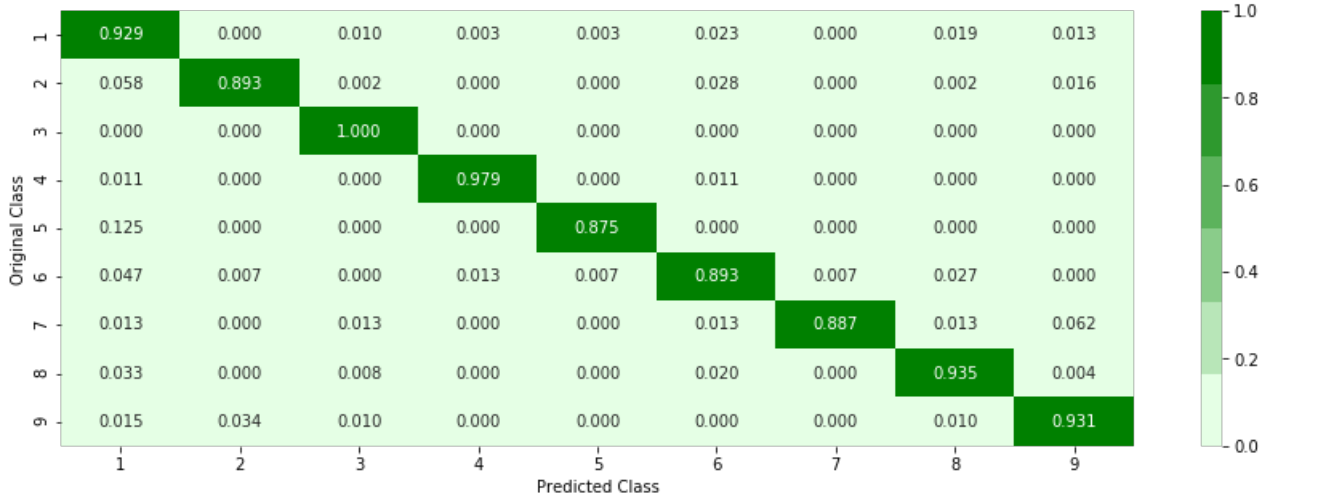

```
For values of best alpha =  3 The train log loss is: 0.11857677462339106
For values of best alpha =  3 The cross validation log loss is: 0.2187440766511307
For values of best alpha =  3 The test log loss is: 0.24465443361017156
Number of misclassified points  6.117755289788408
---------------------------------------------------- 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.]
```

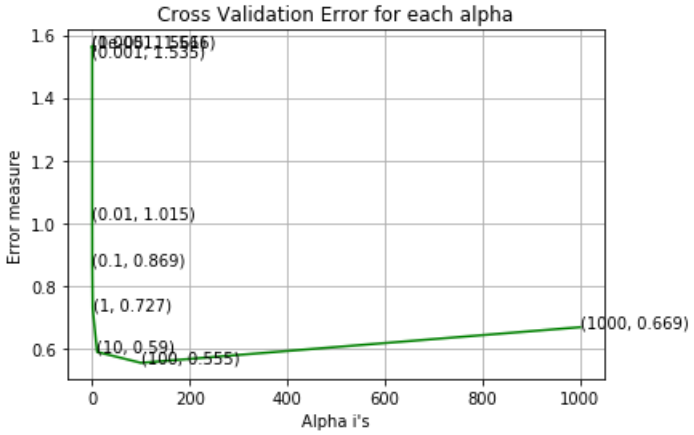
4.1.3. Logistic Regression

In [0]:

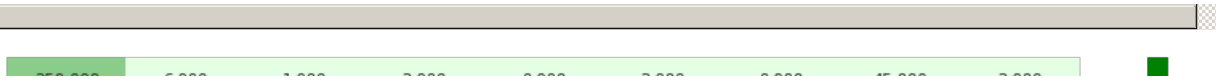
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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

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

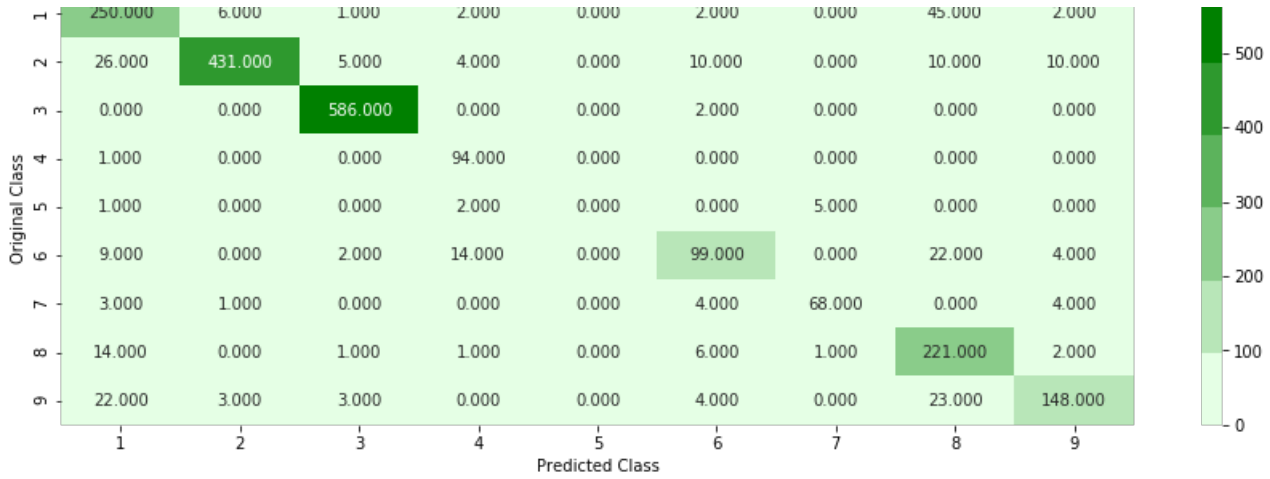
```
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv,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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  1e-05 is 1.5614336766774481
log_loss for c =  0.0001 is 1.5657386530711301
log_loss for c =  0.001 is 1.5350841439080387
log_loss for c =  0.01 is 1.0151976911678842
log_loss for c =  0.1 is 0.868559720610383
log_loss for c =  1 is 0.7272286480032448
log_loss for c =  10 is 0.5896647661148934
log_loss for c =  100 is 0.5548482034769809
log_loss for c =  1000 is 0.6685075026955517
```

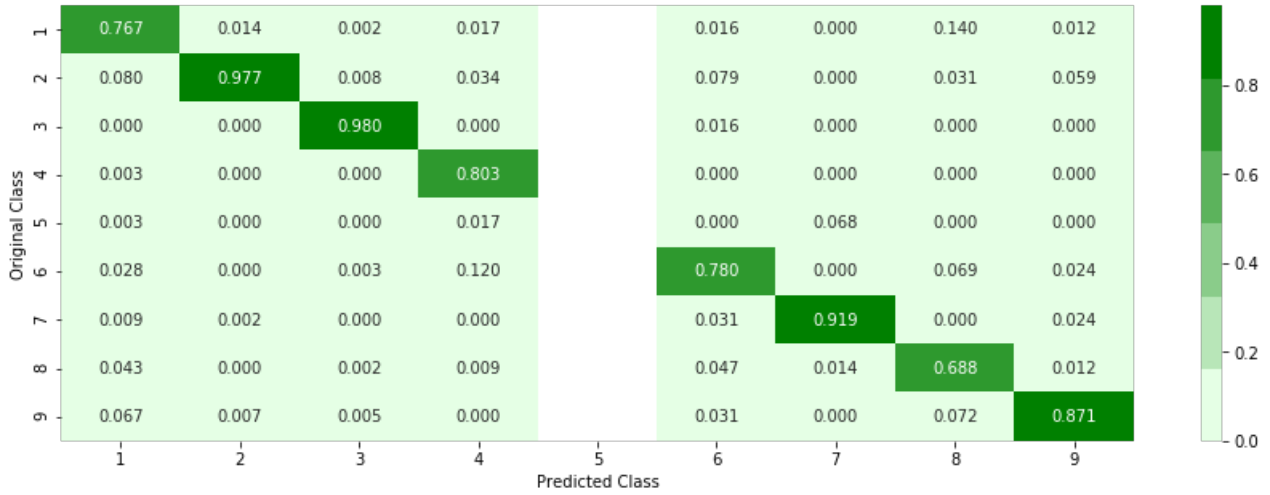



```
log loss for train data 0.5033283230023188
log loss for cv data 0.5548482034769809
log loss for test data 0.5475540572812383
Number of misclassified points  12.741490340386385
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```



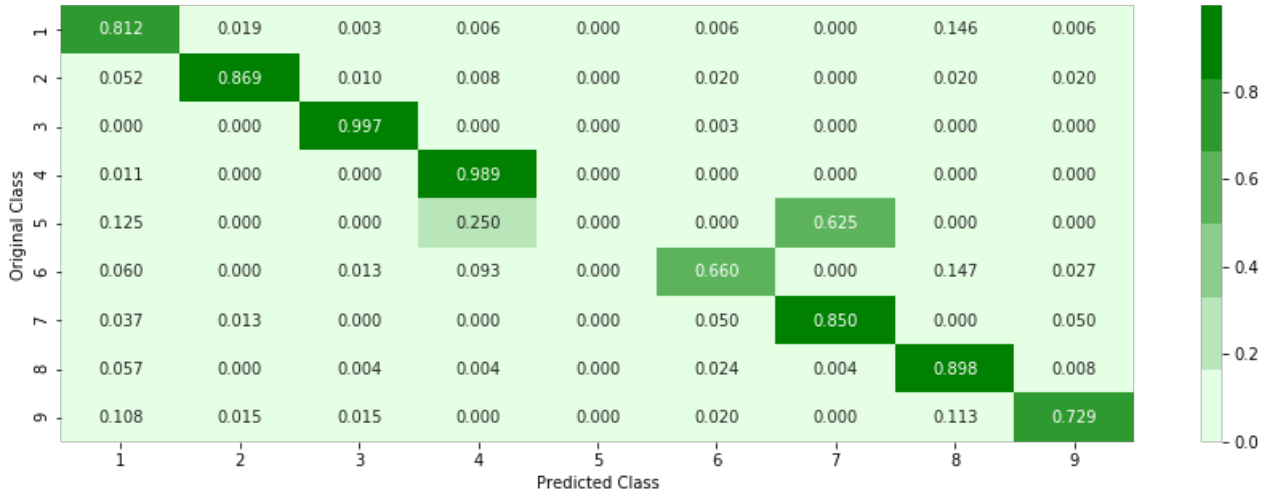


---------- Precision matrix ----------



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.]

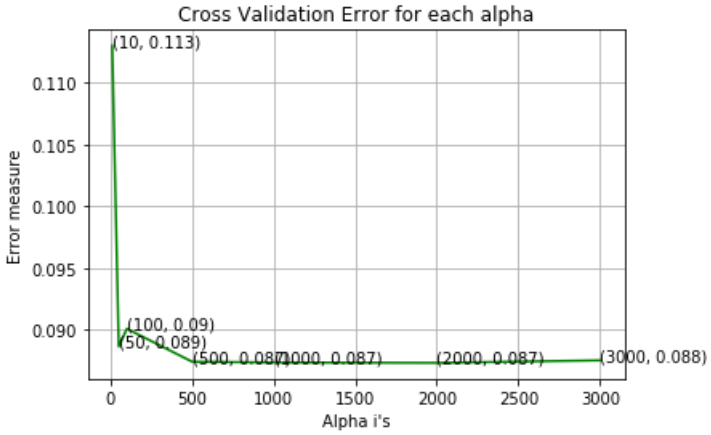
4.1.4. Random Forest Classifier

In [0]:

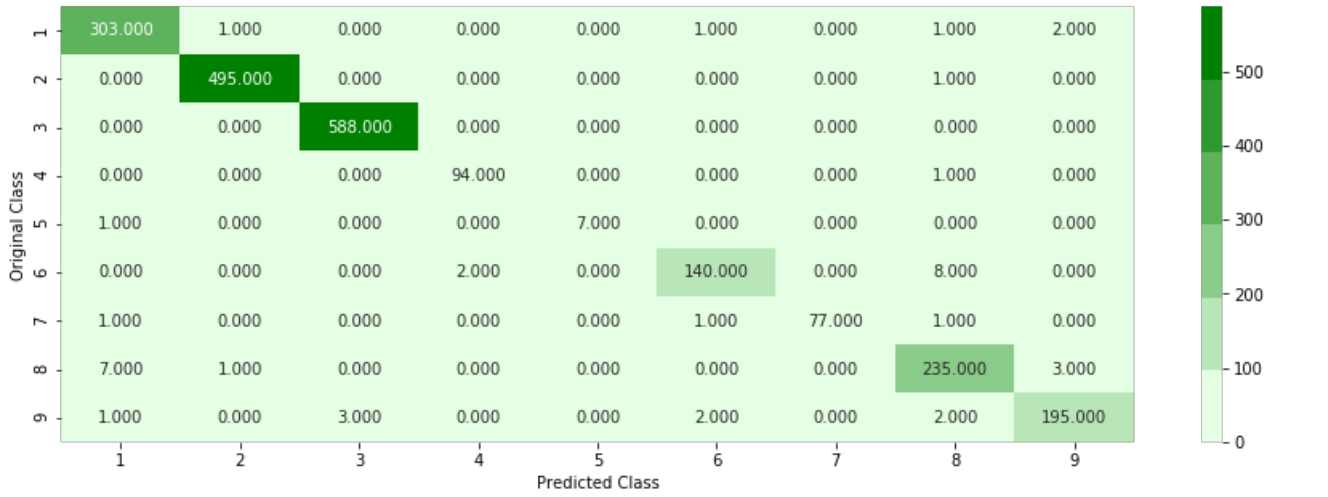
```
# -------------------------------
```

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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

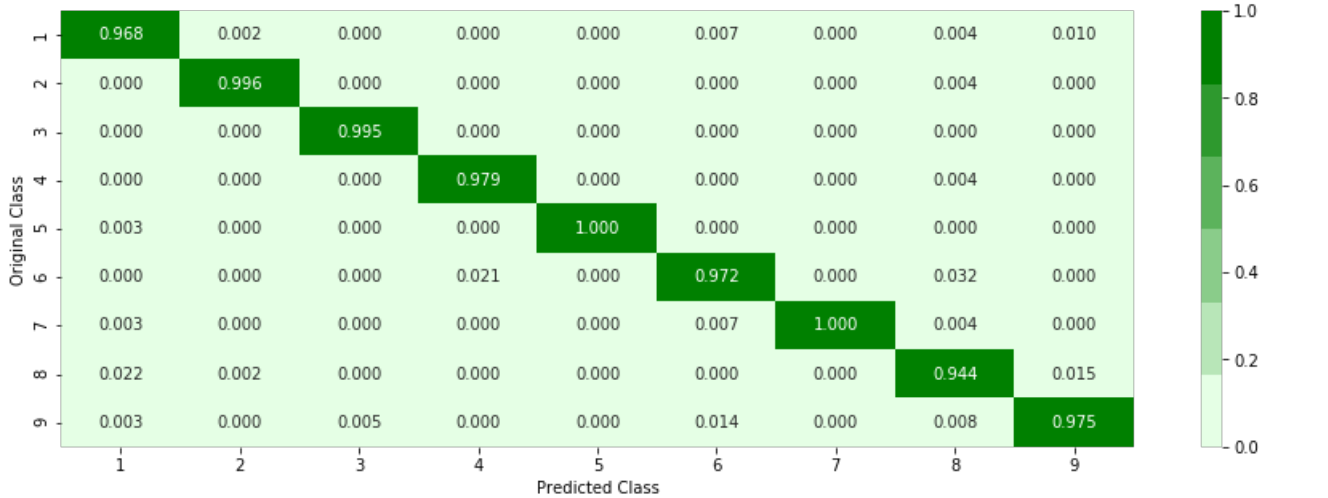
```
log_loss for c =  10 is 0.11296935108083249
log_loss for c =  50 is 0.0886366731716961
log_loss for c =  100 is 0.09007836089870518
log_loss for c =  500 is 0.08741158642668809
log_loss for c =  1000 is 0.08735338601116735
log_loss for c =  2000 is 0.08732039720514327
log_loss for c =  3000 is 0.08752307781182385
```



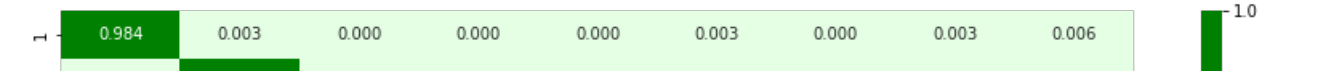

```
For values of best alpha =  2000 The train log loss is: 0.026050507237571532
For values of best alpha =  2000 The cross validation log loss is: 0.08732039720514327
For values of best alpha =  2000 The test log loss is: 0.08338504681236152
Number of misclassified points  1.8399264029438822
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```

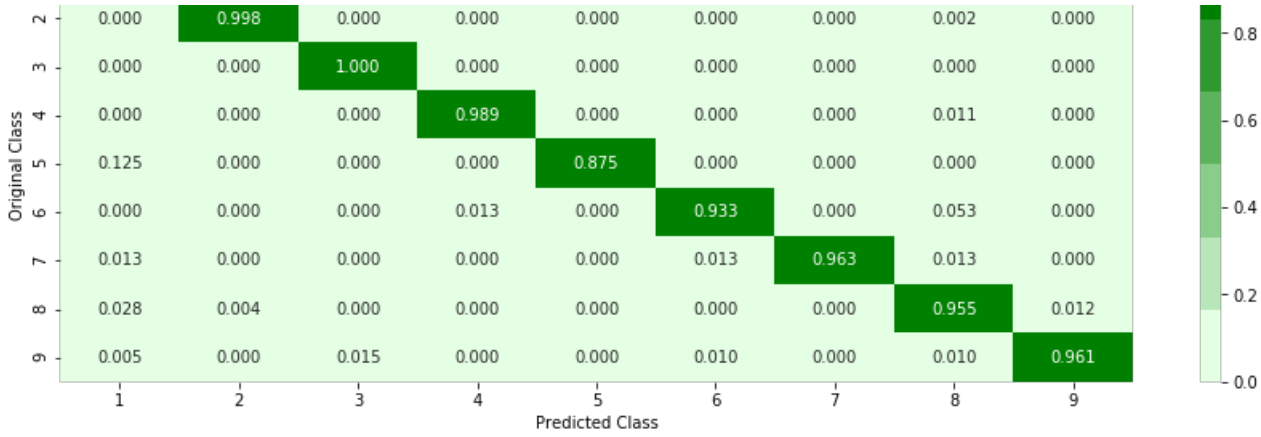



```
---------------------------------------------------- Precision matrix -----------------------------
----------------
```




```
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.]
```

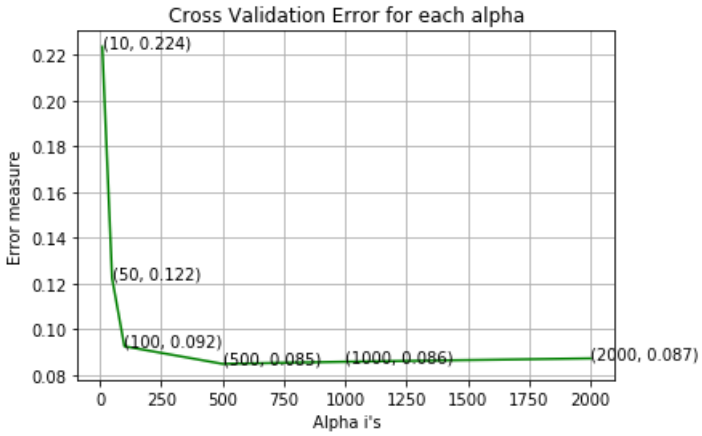
4.1.5. XgBoost Classification

In [0]:

```
# 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, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-
using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

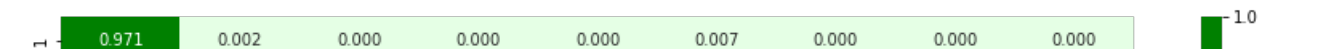
```
plt.show()

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

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

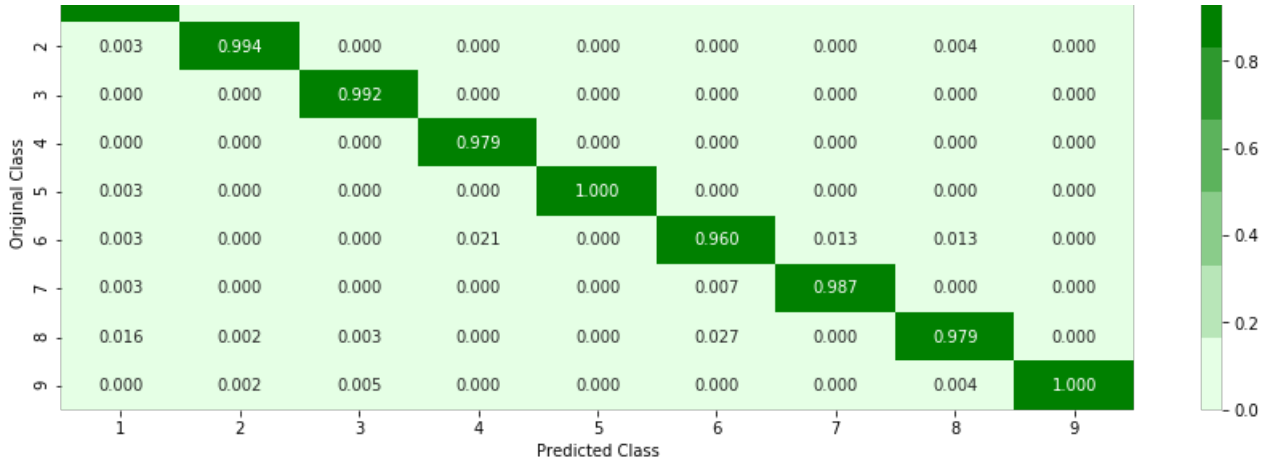
```
log_loss for c =  10 is 0.22357648665306107
log_loss for c =  50 is 0.12219973107348131
log_loss for c =  100 is 0.09247141868319304
log_loss for c =  500 is 0.08478569531409207
log_loss for c =  1000 is 0.08570912356414268
log_loss for c =  2000 is 0.0871734330714657
```



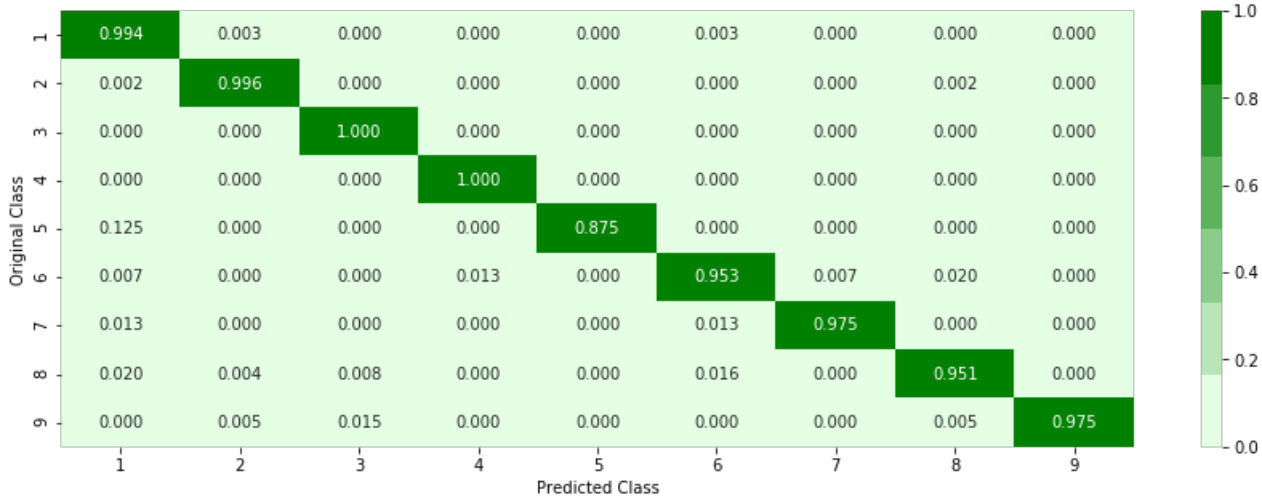

```
For values of best alpha =  500 The train log loss is: 0.02209110745927054
For values of best alpha =  500 The cross validation log loss is: 0.08478569531409207
For values of best alpha =  500 The test log loss is: 0.07021205690395897
Number of misclassified points  1.4259429622815087
---------------------------------------------------- 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.]
```

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [0]:

```
# 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)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  7.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  9.2min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 23.7min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 30.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 44.8min finished
```

Out[0]:

```
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)
```

In [0]:

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

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 5, 'learning_rate': 0.03, 'colsample_bytree':
0.3}
```

In [0]:

```
# 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, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------

x_cfl=XGBClassifier(n_estimators=1000, learning_rate=0.03, colsample_bytree=0.3, max_depth=5)
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 0.020451934563151235
cv loss 0.07975627994347553
```

```
test loss 0.06960455439919733
```

Here we computing byte files Bi-grams vectorizer

4.1.1 Bi-Grams Byte files

In [0]:

```
#intially create five folders #first #second #thrid #fourth #fifth #this code tells us about rando
m 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)
#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 ='sixth'
folder_7 ='seventh'
folder_8 ='eigth'
folder_9 ='ninth'
folder_10 ='tenth'
for i in [folder_1,folder_2,folder_3,folder_4,folder_5,folder_6,folder_7,folder_8,folder_9,folder_1
0]:
    if not os.path.isdir(i):
        os.makedirs(i)

source='byteFiles/'
files = os.listdir('byteFiles')
#ID=df['Id'].tolist()
data=range(0,10868)
#r.shuffle(data)
count=0
for i in tqdm(range(0,10868),position=0):
    if i % 10==0:
        shutil.copy(source+files[data[i]],'first')
    elif i%10==1:
        shutil.copy(source+files[data[i]],'second')
    elif i%10==2:
        shutil.copy(source+files[data[i]],'third')
    elif i%10==3:
        shutil.copy(source+files[data[i]],'fourth')
    elif i%10==4:
        shutil.copy(source+files[data[i]],'fifth')
    elif i%10==5:
        shutil.copy(source+files[data[i]],'sixth')
    elif i%10==6:
        shutil.copy(source+files[data[i]],'seventh')
    elif i%10==7:
        shutil.copy(source+files[data[i]],'eigth')
    elif i%10==8:
        shutil.copy(source+files[data[i]],'ninth')
    elif i%10==9:
        shutil.copy(source+files[data[i]],'tenth')
```

```
100%|██████████| 10868/10868 [34:52<00:00,  5.19it/s]
```

In [0]:

```
#shutil.rmtree('tenth')
```

In [0]:

```
path, dirs, files = next(os.walk("first"))
```

```
file_count = len(files)
print('No of files in folder-1',file_count)

path, dirs, files = next(os.walk("second"))
file_count = len(files)
print('No of files in folder-2',file_count)

path, dirs, files = next(os.walk("third"))
file_count = len(files)
print('No of files in folder-3',file_count)

path, dirs, files = next(os.walk("fourth"))
file_count = len(files)
print('No of files in folder-4',file_count)

path, dirs, files = next(os.walk("fifth"))
file_count = len(files)
print('No of files in folder-5',file_count)

path, dirs, files = next(os.walk("sixth"))
file_count = len(files)
print('No of files in folder-6',file_count)

path, dirs, files = next(os.walk("seventh"))
file_count = len(files)
print('No of files in folder-7',file_count)

path, dirs, files = next(os.walk("eigth"))
file_count = len(files)
print('No of files in folder-8',file_count)

path, dirs, files = next(os.walk("ninth"))
file_count = len(files)
print('No of files in folder-9',file_count)

path, dirs, files = next(os.walk("tenth"))
file_count = len(files)
print('No of files in folder-10',file_count)
```

```
No of files in folder-1 1087
No of files in folder-2 1087
No of files in folder-3 1087
No of files in folder-4 1087
No of files in folder-5 1087
No of files in folder-6 1087
No of files in folder-7 1087
No of files in folder-8 1087
No of files in folder-9 1086
No of files in folder-10 1086
```

In [0]:

```
byte_vocab =
"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,2(
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,
4,45,46,47,48,49,4a,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,(
,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
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,
b,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,(
,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
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [0]:

```
byte_bigram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for j in range(0, len(byte_vocab.split(','))):
        byte_bigram_vocab.append(v + ' ' +byte_vocab.split(',')[j])
len(byte_bigram_vocab)
```

Out[0]:

```
66049
```

In [0]:

```
byte_bigram_vocab[:5]
```

Out[0]:

```
['00 00', '00 01', '00 02', '00 03', '00 04']
```

In [0]:

```
#for removing a  directory and its files
#shutil.rmtree('asm_image')
```

In [0]:

```
%%time
from tqdm import tqdm
import scipy
from sklearn.feature_extraction.text import CountVectorizer
def firstprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect1 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('first')),position=0):
        f = open('first/' + file)
        byte_bigram_vect1[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect1.npz', byte_bigram_vect1)

def secondprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect2 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('second')),position=0):
        f = open('second/' + file)
        byte_bigram_vect2[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect2.npz', byte_bigram_vect2)

def thirdprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect3 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('third')),position=0):
        f = open('third/' + file)
        byte_bigram_vect3[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect3.npz', byte_bigram_vect3)

def fourthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect4= scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('fourth')),position=0):
        f = open('fourth/' + file)
        byte_bigram_vect4[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect4.npz', byte_bigram_vect4)

def fifthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect5 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('fifth')),position=0):
        f = open('fifth/' + file)
        byte_bigram_vect5[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect5.npz', byte_bigram_vect5)

def sixthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect6 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('sixth')),position=0):
        f = open('sixth/' + file)
```

```
        byte_bigram_vect6[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect6.npz', byte_bigram_vect6)

def seventhprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect7 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('seventh')),position=0):
        f = open('seventh/' + file)
        byte_bigram_vect7[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect7.npz', byte_bigram_vect7)

def eigthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect8 = scipy.sparse.csr_matrix((1087, 66049))
    for i, file in tqdm(enumerate(os.listdir('eigth')),position=0):
        f = open('eigth/' + file)
        byte_bigram_vect8[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect8.npz', byte_bigram_vect8)

def ninthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect9 = scipy.sparse.csr_matrix((1086, 66049))
    for i, file in tqdm(enumerate(os.listdir('ninth')),position=0):
        f = open('ninth/' + file)
        byte_bigram_vect9[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect9.npz', byte_bigram_vect9)

def tenthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_vect10 = scipy.sparse.csr_matrix((1086, 66049))
    for i, file in tqdm(enumerate(os.listdir('tenth')),position=0):
        f = open('tenth/' + file)
        byte_bigram_vect10[i, :]+=
scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('byte_bigram_vect10.npz', byte_bigram_vect10)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 162 µs
```

In [0]:

```
%%time
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)
    p6=Process(target=sixthprocess)
    p7=Process(target=seventhprocess)
    p8=Process(target=eigthprocess)
    p9=Process(target=ninthprocess)
    p10=Process(target=tenthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    p6.start()
    p7.start()
    p8.start()
    p9.start()
```

```
    p9.start()
    p10.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()
    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()


if __name__=="__main__":
    main()
```

```
1087it [1:30:48,  5.01s/it]
1086it [1:31:53,  5.08s/it]
1087it [1:32:45,  5.12s/it]
1087it [1:33:07,  5.14s/it]
1087it [1:33:50,  5.18s/it]
1087it [1:34:06,  5.19s/it]
1086it [1:34:39,  5.23s/it]
1087it [1:35:17,  5.26s/it]
1087it [1:35:38,  5.28s/it]
1087it [1:36:06,  5.31s/it]
```

```
CPU times: user 8.65 s, sys: 4.96 s, total: 13.6 s
Wall time: 1h 36min 51s
```

In [0]:

```
import scipy
byte_bigram_vect1=scipy.sparse.load_npz('byte_bigram_vect1.npz')
byte_bigram_vect2=scipy.sparse.load_npz('byte_bigram_vect2.npz')
byte_bigram_vect3=scipy.sparse.load_npz('byte_bigram_vect3.npz')
byte_bigram_vect4=scipy.sparse.load_npz('byte_bigram_vect4.npz')
byte_bigram_vect5=scipy.sparse.load_npz('byte_bigram_vect5.npz')
byte_bigram_vect6=scipy.sparse.load_npz('byte_bigram_vect6.npz')
byte_bigram_vect7=scipy.sparse.load_npz('byte_bigram_vect7.npz')
byte_bigram_vect8=scipy.sparse.load_npz('byte_bigram_vect8.npz')
byte_bigram_vect9=scipy.sparse.load_npz('byte_bigram_vect9.npz')
byte_bigram_vect10=scipy.sparse.load_npz('byte_bigram_vect10.npz')
```

In [0]:

```
byte_bi_grams =
scipy.sparse.vstack((byte_bigram_vect1,byte_bigram_vect2,byte_bigram_vect3,byte_bigram_vect4,byte_
bigram_vect5,byte_bigram_vect6,byte_bigram_vect7,byte_bigram_vect8,byte_bigram_vect9,byte_bigram_ve
ct10)).tocsr()
```

In [0]:

```
byte_bi_grams
```

Out[0]:

```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'
 with 495857441 stored elements in Compressed Sparse Row format>
```

In [0]:

```
from sklearn.preprocessing import normalize
final_byte_bigram = normalize(byte_bi_grams)
```

In [0]:

```
final_byte_bigram
```

Out[0]:

```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'
 with 495857441 stored elements in Compressed Sparse Row format>
```

In [0]:

```
scipy.sparse.save_npz('final_byte_bigram.npz', final_byte_bigram)
```

Below we mapping class labels to our Bi-Gram vectorizers using ID we will using this at end

In [0]:

```
#file sizes of byte files

files=os.listdir('first')
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('first/'+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_1=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('second')
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('second/'+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_2=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('third')
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('third/'+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_3=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('fourth')
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('fourth/'+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_4=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('fifth')
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('fifth/'+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_5=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('sixth')
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('sixth/'+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_6=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('seventh')
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('seventh/'+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_7=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('eigth')
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('eigth/'+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_8=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('ninth')
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('ninth/'+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_9=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})

files=os.listdir('tenth')
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('tenth/'+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_10=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
```

In [0]:

```
byte_fnames = [data_size_byte_1,data_size_byte_2,data_size_byte_3,data_size_byte_4,data_size_byte_5
,data_size_byte_6,data_size_byte_7,data_size_byte_8,data_size_byte_9,data_size_byte_10]
```

In [0]:

```
byte_result_class = pd.concat(byte_fnames)
```

In [0]:

```
byte_result_class = pd.DataFrame(byte_result_class)
byte_result_class.head()
```

Out[0]:

	Class	ID	size
0	8	gCQ70meuzrYAFaWDxZJv	2.024902
1	3	DZSwtHBVTqhivJscaoWA	8.099609
2	3	1u3qTGiRvckQZW7dBY58	8.099609
3	6	HD3SgIFw48AUEILe57oi	0.552246
4	1	bW4NY5lZng7LJeDjpQSK	0.991211

In [0]:

```
byte_result_class.to_csv('byte_result_class.csv')
```

In [0]:

```
byte_result_class = pd.read_csv('byte_result_class.csv',index_col=0)
byte_result_class.head()
```

Out[0]:

	Class	ID	size
0	8	gCQ70meuzrYAFaWDxZJv	2.024902
1	3	DZSwtHBVTqhivJscaoWA	8.099609
2	3	1u3qTGiRvckQZW7dBY58	8.099609
3	6	HD3SgIFw48AUEILe57oi	0.552246
4	1	bW4NY5lZng7LJeDjpQSK	0.991211

In [0]:

```
y = byte_result_class['Class']
```

In [0]:

```
final_byte_bigram=final_byte_bigram.todense()
```

In [0]:

```
from sklearn.preprocessing import normalize
final_byte_bigrams = pd.DataFrame(final_byte_bigram,columns=byte_bigram_vocab).fillna(0)
#final_opcode_bigram=final_opcode_bigram.to_dense()
final_byte_bigrams.head()
```

Out[0]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	...	?? f7	?? f8	?? f9	?? fa	?
0	0.993389	0.011028	0.006807	0.006370	0.007566	0.005482	0.005019	0.005379	0.005096	0.004684	...	0.0	0.0	0.0	0.0	0
1	0.811853	0.006804	0.003254	0.011241	0.003846	0.008431	0.002514	0.006360	0.004289	0.007395	...	0.0	0.0	0.0	0.0	0
2	0.836139	0.004827	0.003448	0.010481	0.002344	0.007309	0.003448	0.005516	0.002620	0.006620	...	0.0	0.0	0.0	0.0	0
3	0.931650	0.002107	0.000421	0.000421	0.000948	0.000105	0.000211	0.000211	0.000527	0.000316	...	0.0	0.0	0.0	0.0	0
4	0.978322	0.010135	0.004306	0.004080	0.004144	0.001263	0.000648	0.002007	0.001813	0.003335	...	0.0	0.0	0.0	0.0	0

5 rows × 66049 columns

Final Byte files Bi-gram vectorizer

In [0]:

```
final_byte_bigrams.to_csv('final_byte_bigrams.csv')
```

In [0]:

```
new_cols_0 = np.hstack((byte_result_class.columns,final_byte_bigrams.columns))
```

In [0]:

```
final_byte_bigrams = np.hstack((byte_result_class,final_byte_bigrams))
final_byte_bigrams = pd.DataFrame(final_opcode_bigrams,columns=new_cols_0).fillna(0)
final_byte_bigrams.head()
```

In [0]:

```
final_byte_bigrams.to_csv('final_byte_bigrams.csv')
```

4.2 Modeling with .asm files

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

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

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

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**

In [0]:

```
#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')
```

In [0]:

```
#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','movz
x']
    #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']
```

```
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
        with codecs.open('first/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                # https://www.tutorialspoint.com/python3/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


#same as above
def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
```

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

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    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()
```

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


def fifthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    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()
```

In [0]:

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

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2

	ID	HEADER:	text:	Pav:	idata:	data:	bss:	rdata:	edata:	rsrc:	...	edx	esi	eax	ebx	ecx
4	46OZzdsSKDCFV8h7XWXf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [0]:

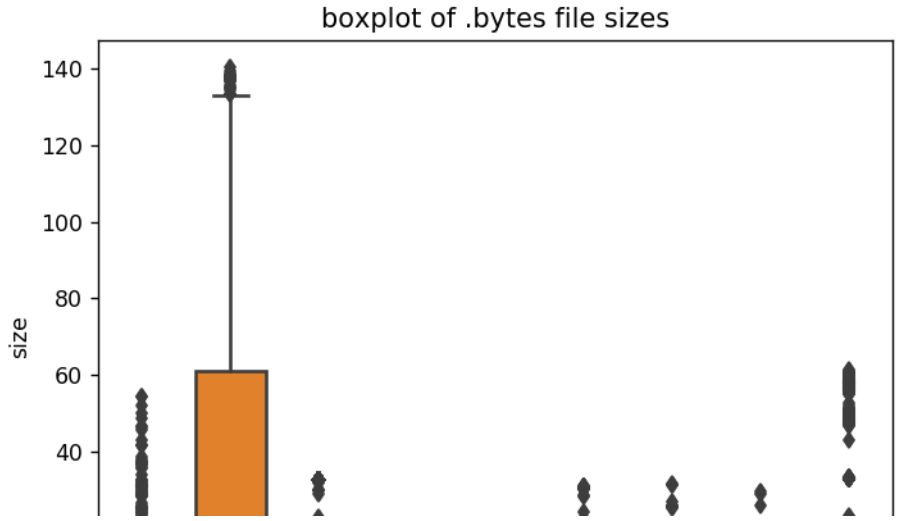
```
#file sizes of byte files

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

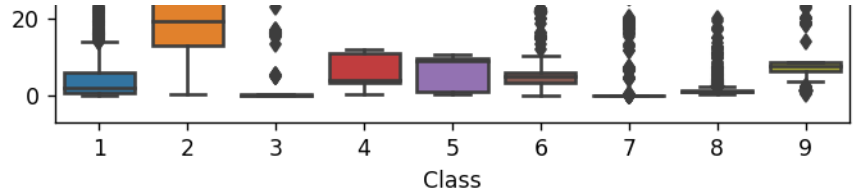
```
   Class                       ID       size
0      9  01azqd4InC7m9JpocGv5  56.229886
1      2  01IsoiSMh5gxyDYTl4CB  13.999378
2      9  01jsnpXSAlgw6aPeDxrU   8.507785
3      1  01kcPWA9K2BOxQeS5Rju   0.078190
4      8  01SuzwMJEIXsK7A8dQbl   0.996723
```

4.2.1.2 Distribution of .asm file sizes

In [0]:

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





In [0]:

```
# 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)
```

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	e
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	1
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	1
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	1
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	1

5 rows × 54 columns

In [0]:

```
dfasm=pd.read_csv("asmoutputfile.csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='left')
result_asm.head()
```

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83	
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14	
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	

5 rows × 53 columns

In [0]:

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

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000343
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000343

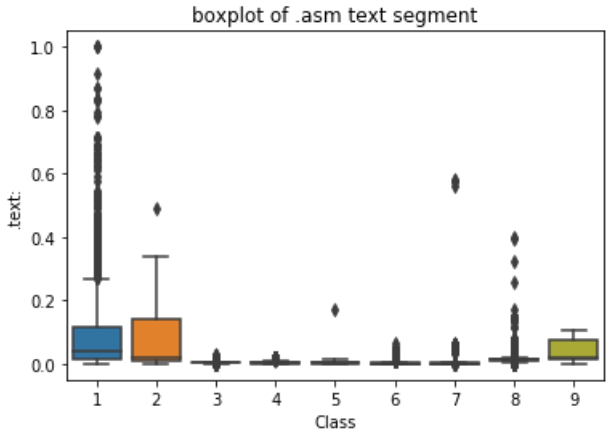
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	.edx
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000248
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000114
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000229

5 rows × 53 columns

4.2.2 Univariate analysis on asm file features

In [0]:

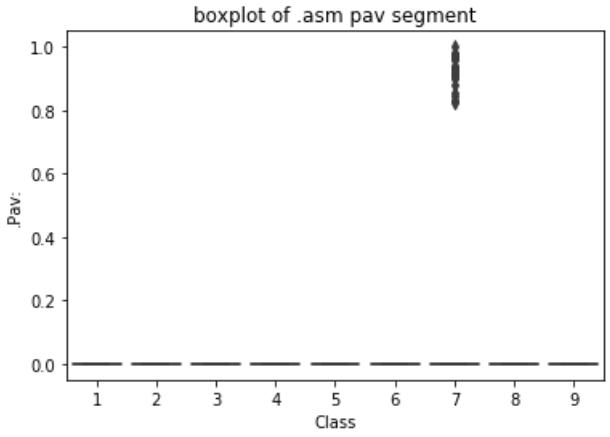
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```



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

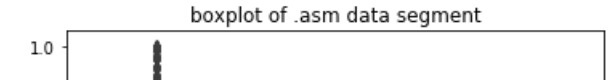
In [0]:

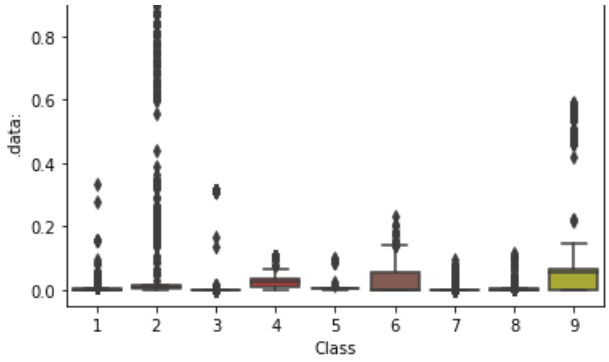
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```



In [0]:

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

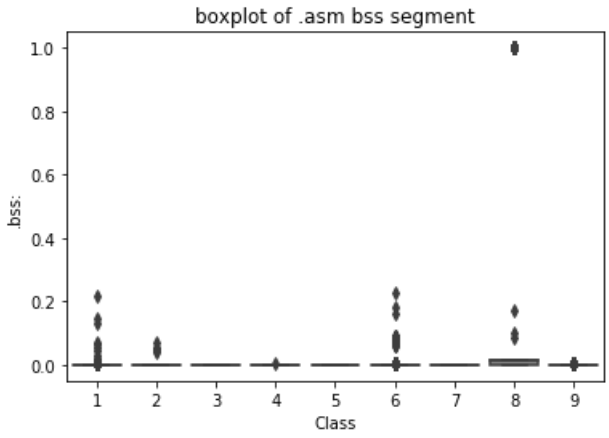




The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [0]:

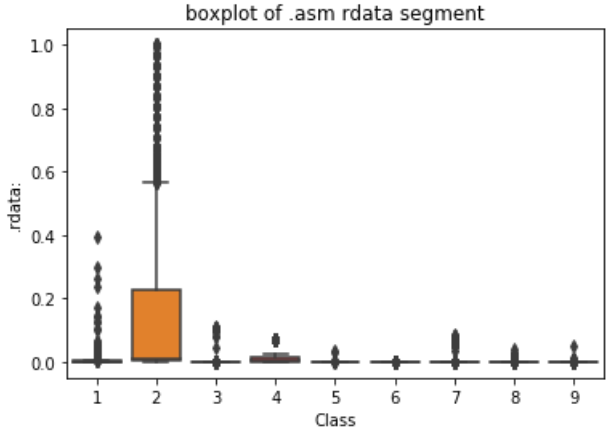
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [0]:

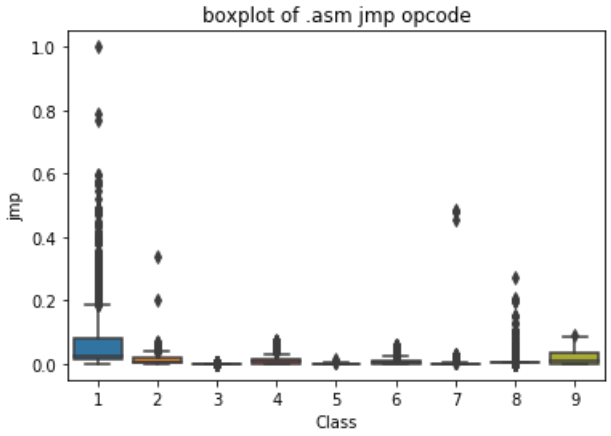
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

In [0]:

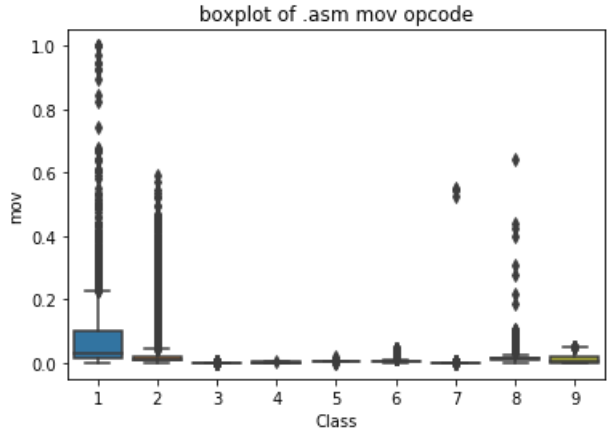
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



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

In [0]:

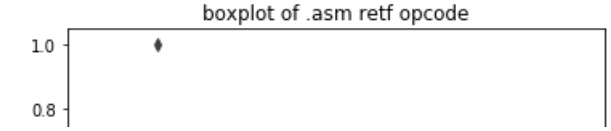
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```

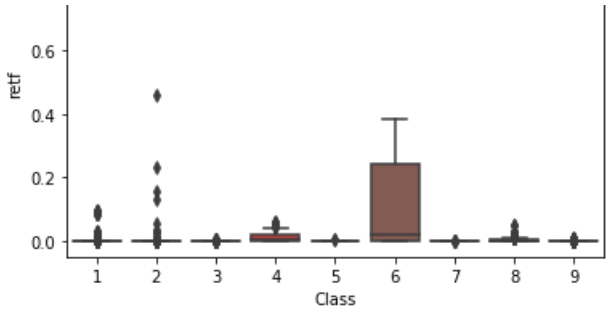


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [0]:

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

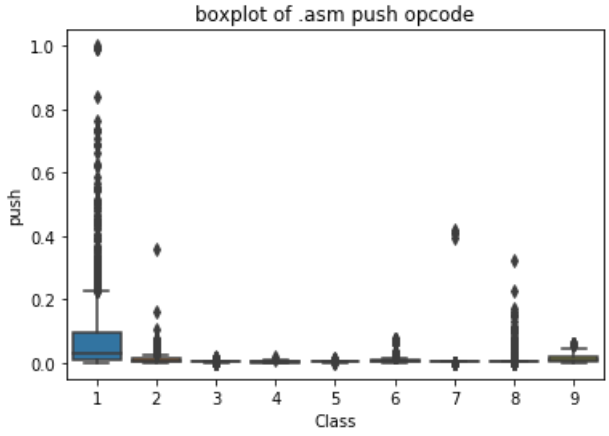




plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [0]:

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

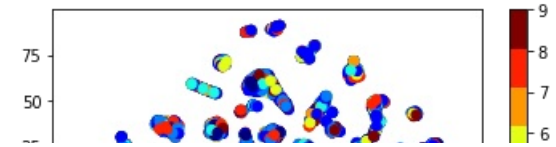


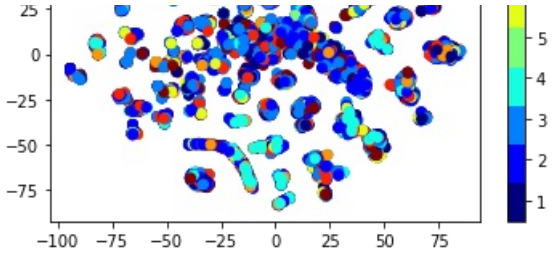
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

In [0]:

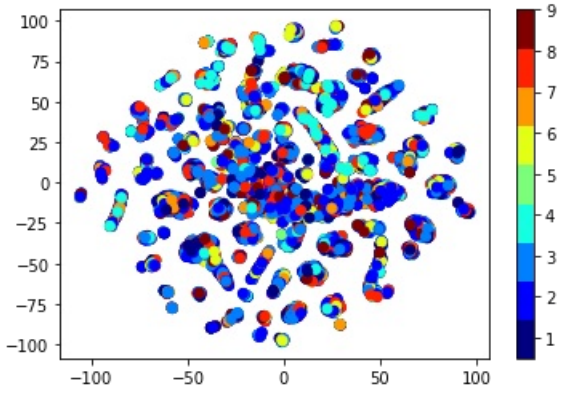
```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic
-neighbourhood-embeddingt-sne-part-1/

#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()
```





In [0]:

```
# by univariate analysis on the .asm file features we are getting very negligible information from

# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing tho
se features
# the plot looks very messy

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



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [0]:

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

In [0]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,tes
t_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y
 train asm,test size=0.20)
```

In [0]:

```
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
dtype: bool
```

4.4. Machine Learning models on features of .asm files

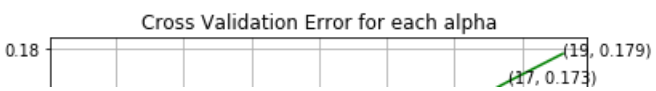
4.4.1 K-Nearest Neigbors

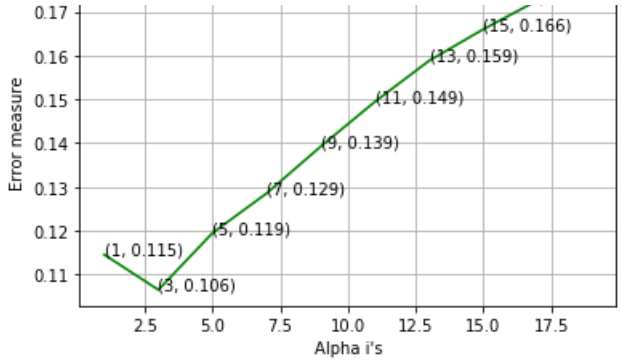
In [0]:

```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.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
```

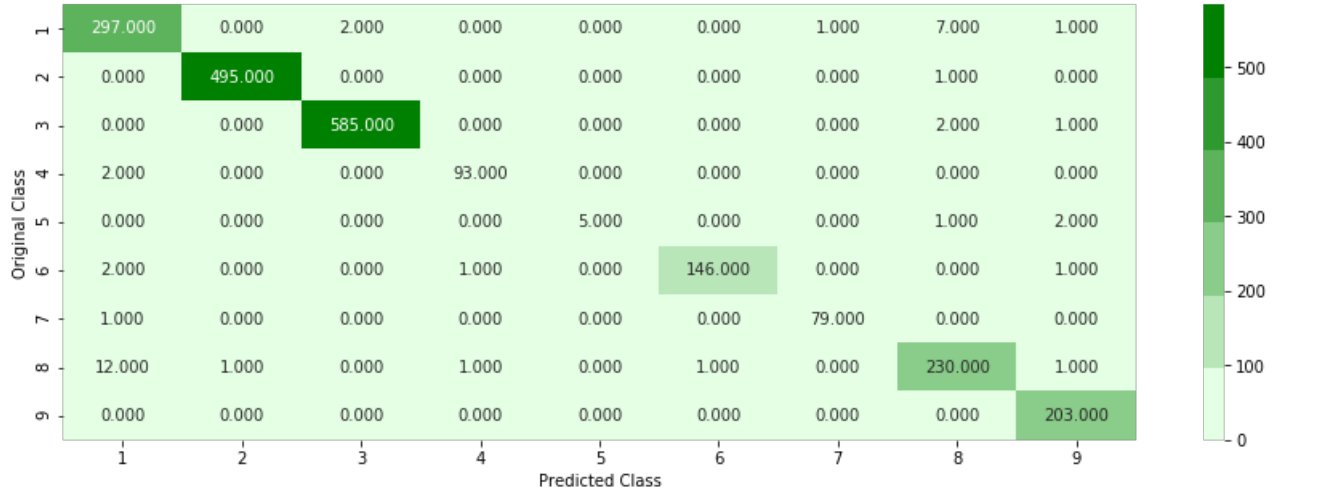
```
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.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, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

```
log_loss for k =  1 is 0.11454442408810245
log_loss for k =  3 is 0.10649397191344541
log_loss for k =  5 is 0.11948036182463083
log_loss for k =  7 is 0.1285551774517421
log_loss for k =  9 is 0.13926172744049972
log_loss for k =  11 is 0.14947917563066215
log_loss for k =  13 is 0.15883712986562093
log_loss for k =  15 is 0.16601633254675957
log_loss for k =  17 is 0.17259959162948854
log_loss for k =  19 is 0.17891115259299978
```

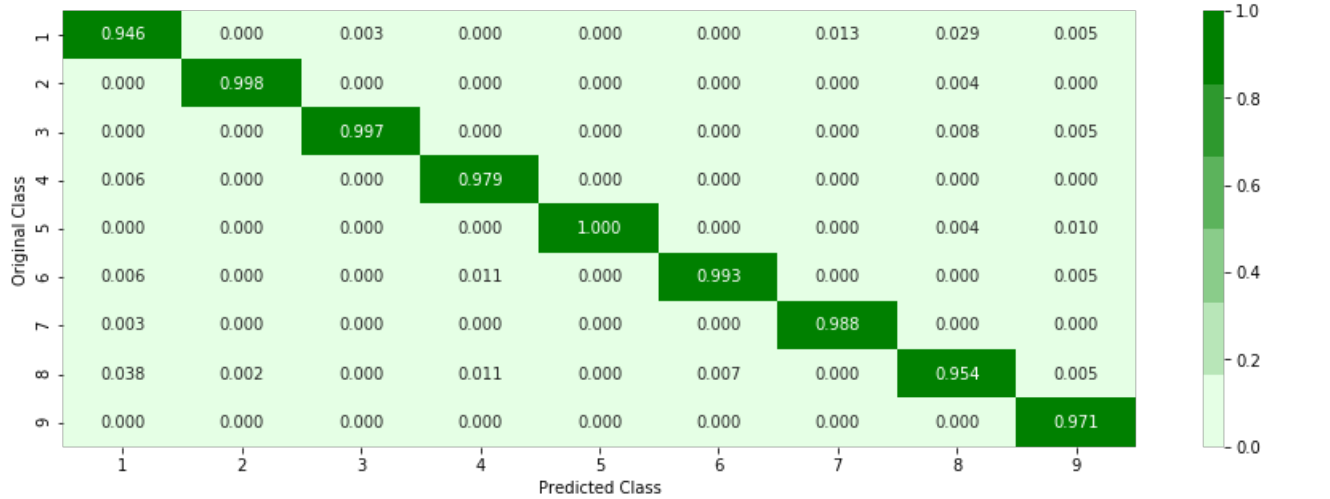




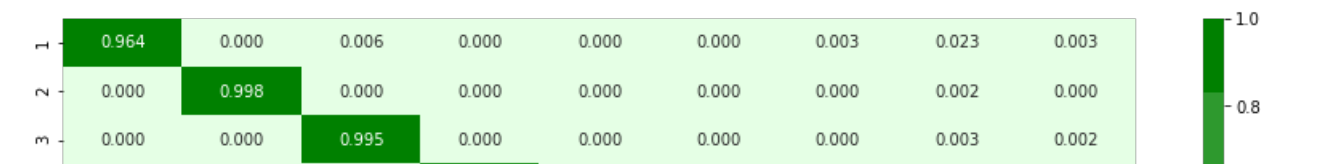

```
log loss for train data 0.06005770679790898
log loss for cv data 0.10649397191344541
log loss for test data 0.07538458958750106
Number of misclassified points  1.8859245630174795
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```

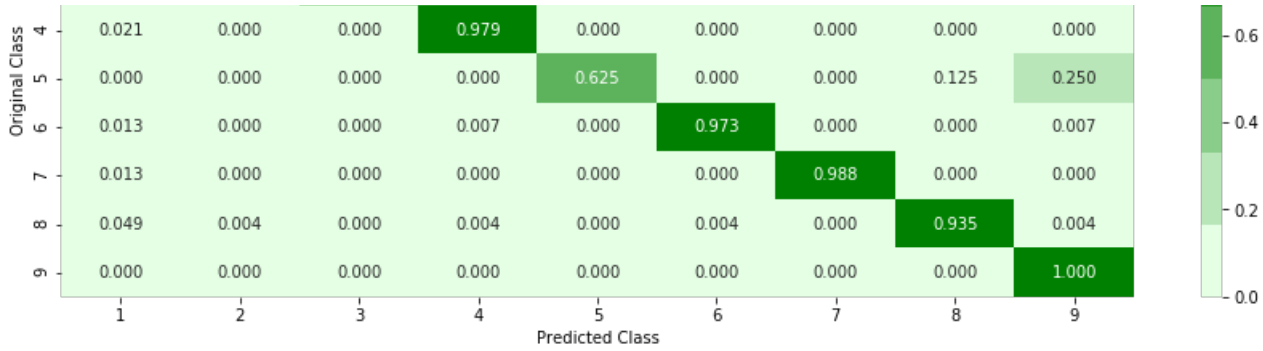



```
---------------------------------------------------- Precision matrix -----------------------------
----------------
```




```
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.]
```

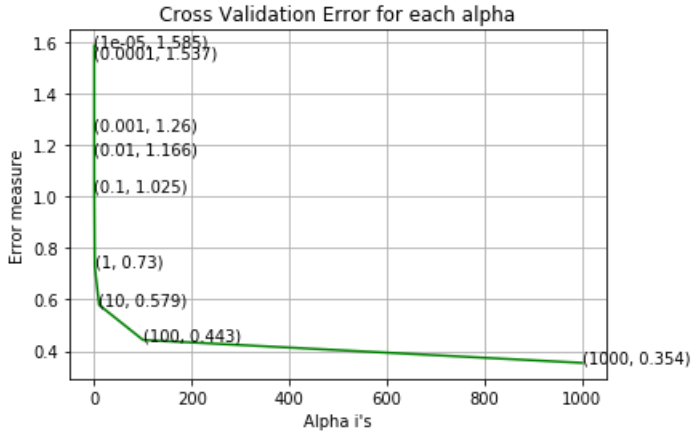
4.4.2 Logistic Regression

In [0]:

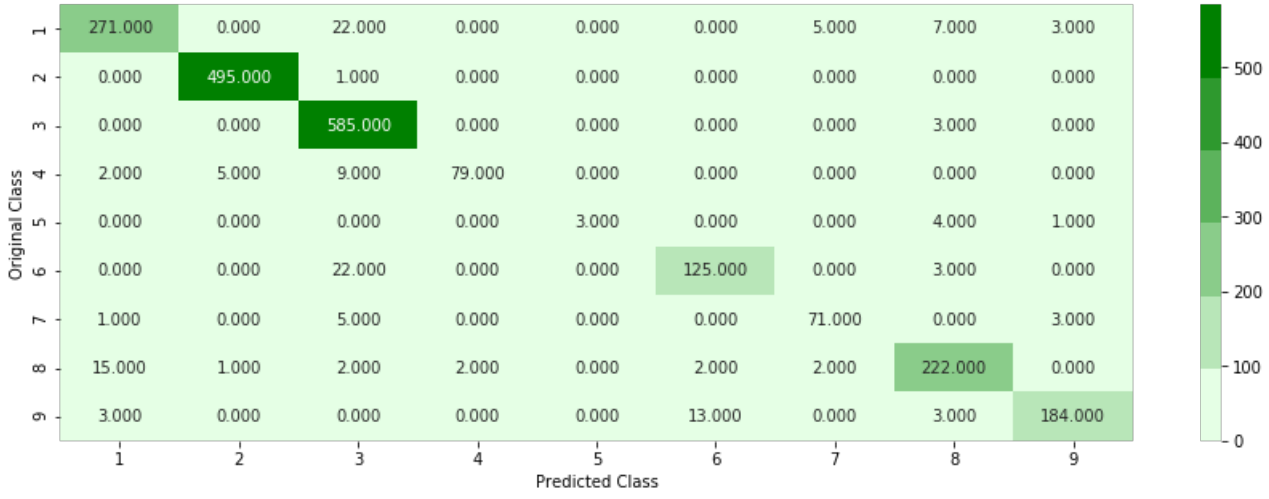
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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


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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

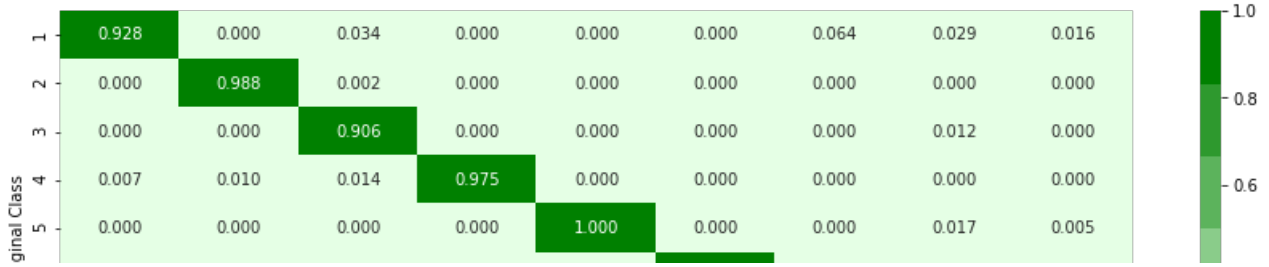
```
log_loss for c =  1e-05 is 1.5851112264695626
log_loss for c =  0.0001 is 1.5365281866220966
log_loss for c =  0.001 is 1.2595660583645187
log_loss for c =  0.01 is 1.1655316925680663
log_loss for c =  0.1 is 1.0251065755242161
log_loss for c =  1 is 0.7299887793841378
log_loss for c =  10 is 0.5791396784600897
log_loss for c =  100 is 0.4434063957513164
log_loss for c =  1000 is 0.35443185391028975
```

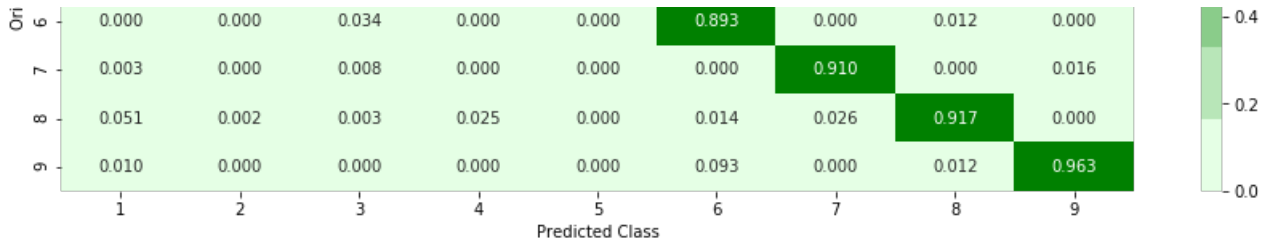



```
log loss for train data 0.32579883559415423
log loss for cv data 0.35443185391028975
log loss for test data 0.33185555092282537
Number of misclassified points  6.39374425022999
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```

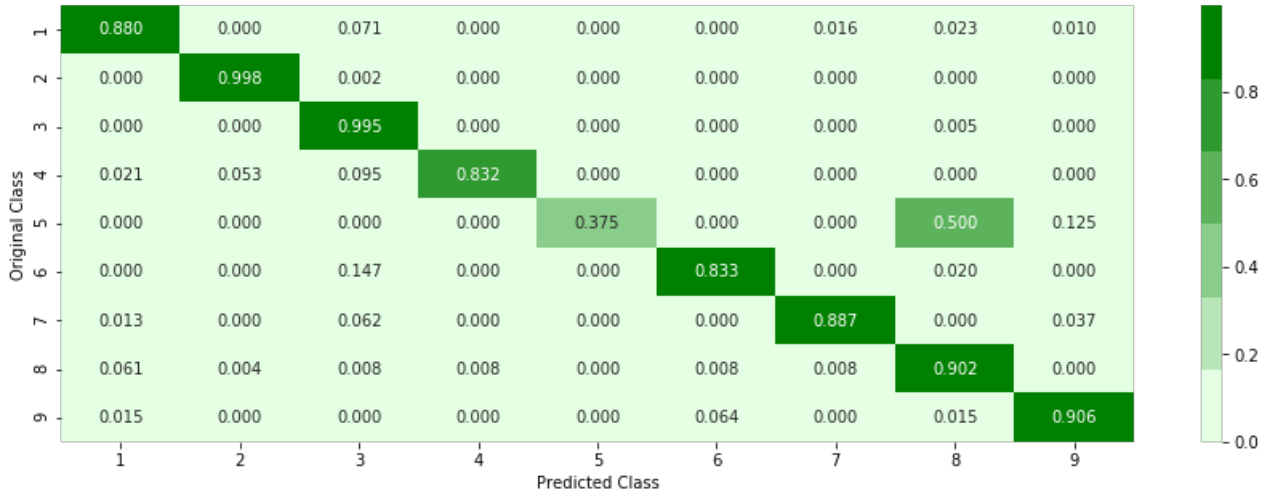



```
---------------------------------------------------- Precision matrix -----------------------------
----------------
```






```
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.]
```

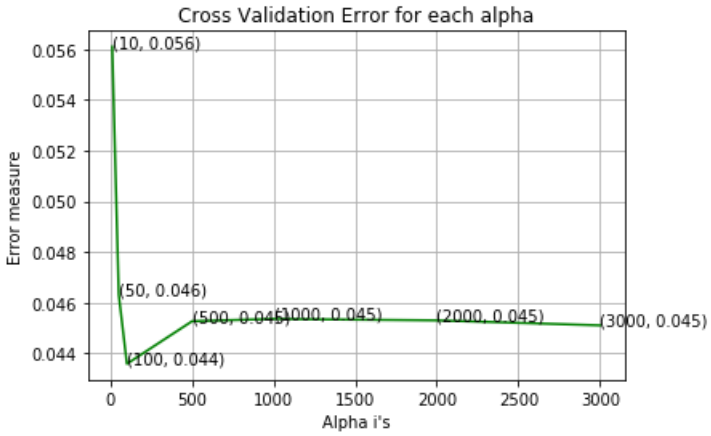
4.4.3 Random Forest Classifier

In [0]:

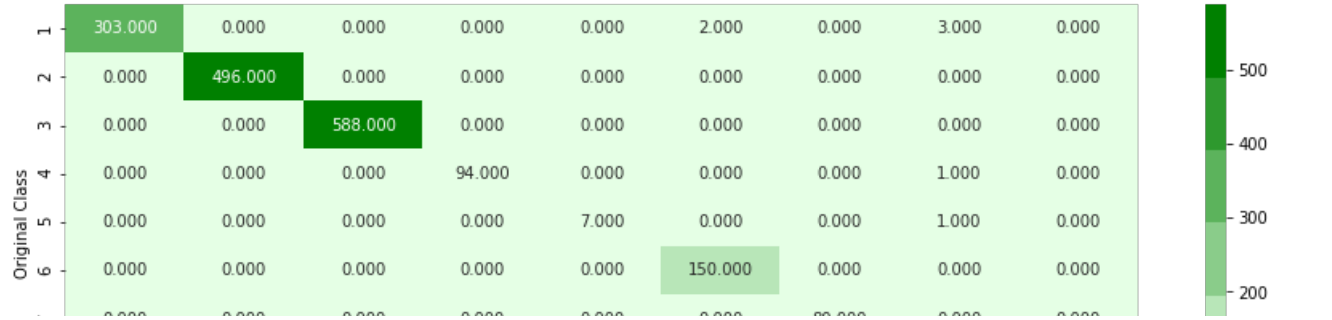
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_, eps=1e-15))
```

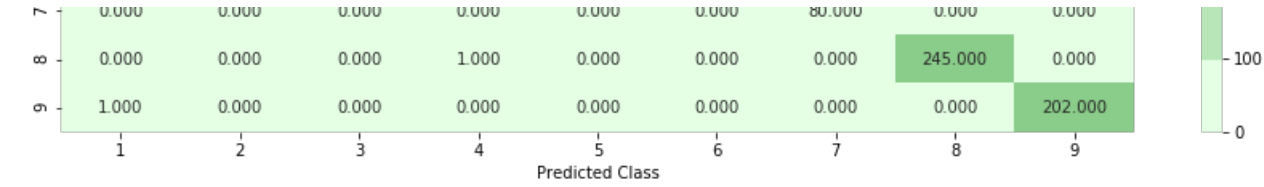
```
for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

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

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

```
log_loss for c =  10 is 0.05609787598646633
log_loss for c =  50 is 0.04631314638193802
log_loss for c =  100 is 0.04357988246021439
log_loss for c =  500 is 0.045264348040874966
log_loss for c =  1000 is 0.04535337252679753
log_loss for c =  2000 is 0.045296299645421464
log_loss for c =  3000 is 0.0450997012228958
```

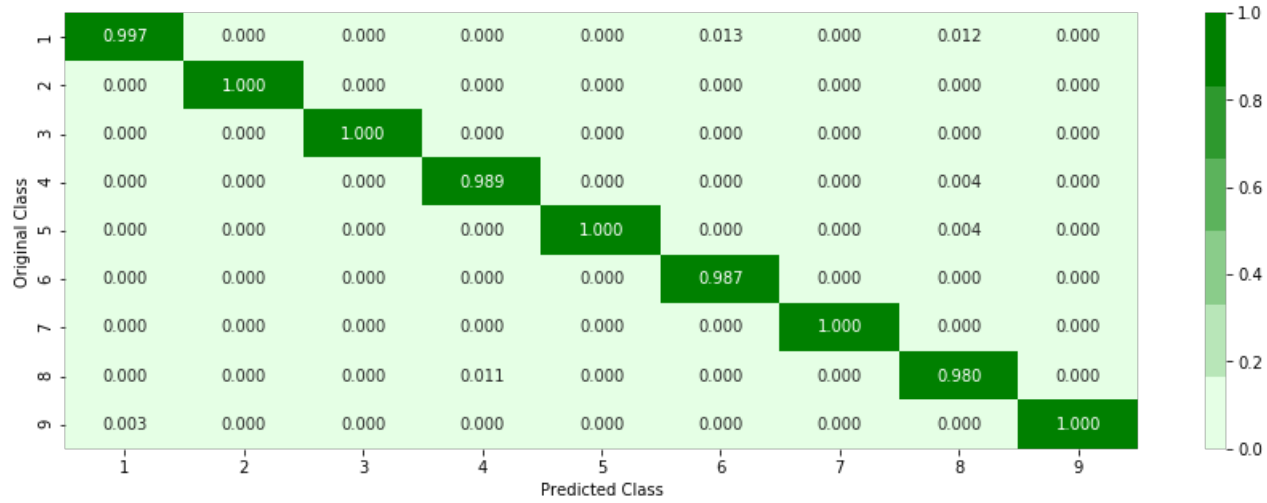



```
log loss for train data 0.021743803044961944
log loss for cv data 0.043579882460214394
log loss for test data 0.023363888255722637
Number of misclassified points  0.41398344066237347
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```



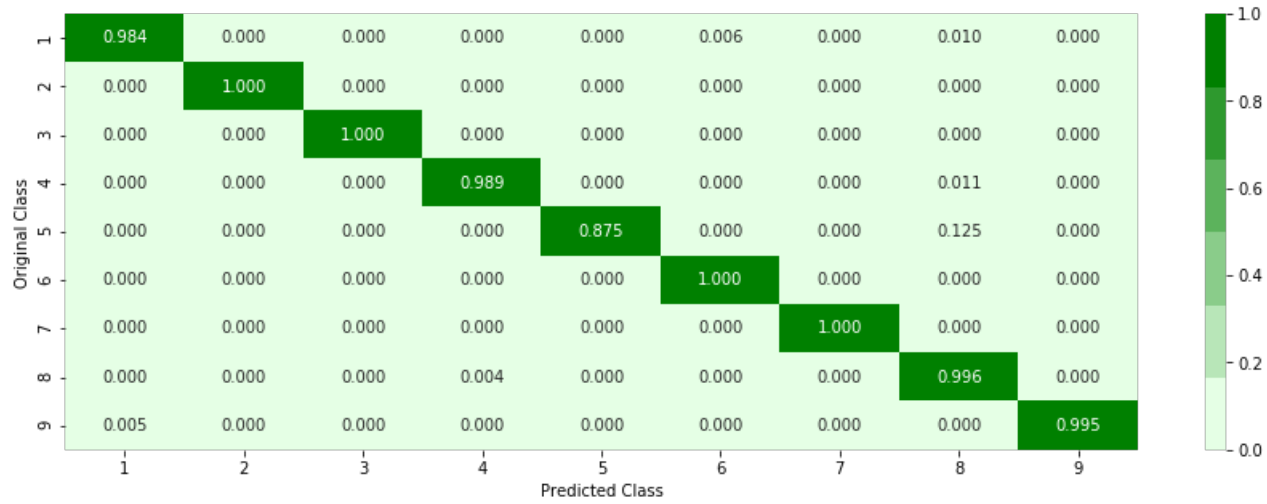


---------- Precision matrix ----------



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.]

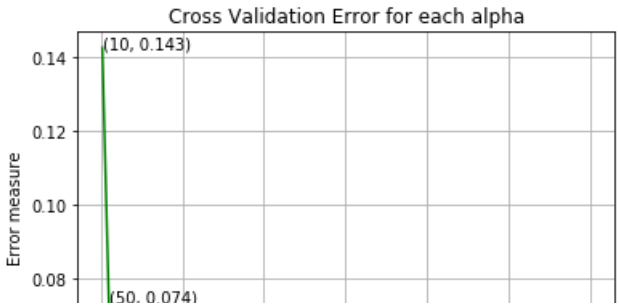
4.4.4 XgBoost Classifier

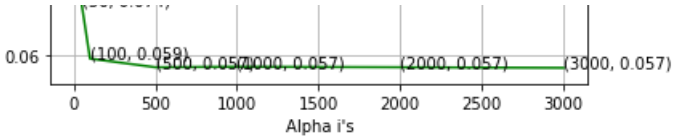
In [0]:

```
# 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)
```

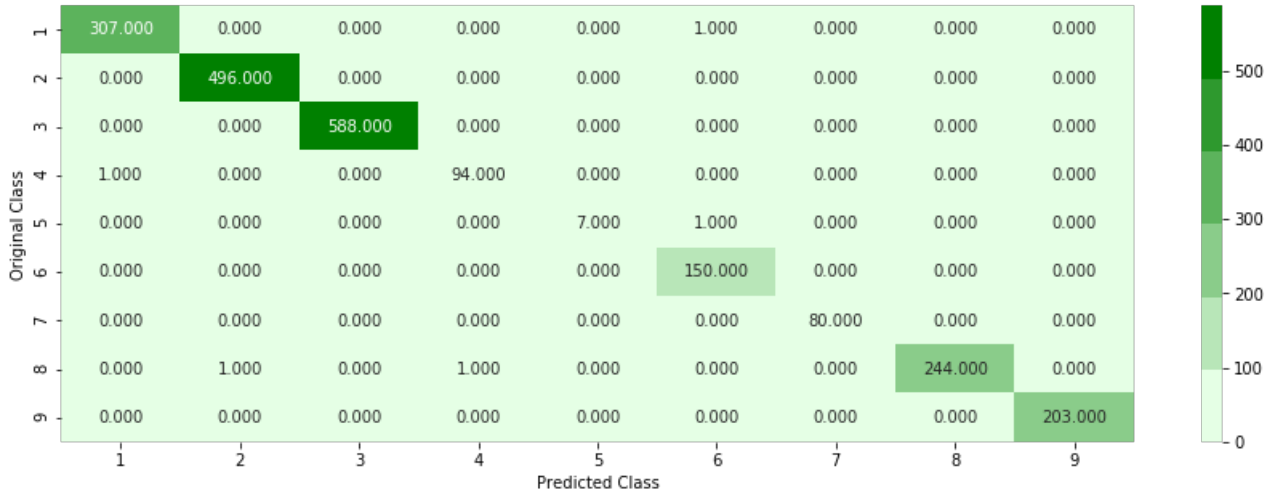
```
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

```
log_loss for c =  10 is 0.1427093302988586
log_loss for c =  50 is 0.07371835909530298
log_loss for c =  100 is 0.05923526939307936
log_loss for c =  500 is 0.05688902001619664
log_loss for c =  1000 is 0.057022612419843445
log_loss for c =  2000 is 0.05689175208182282
log_loss for c =  3000 is 0.05672373738554634
```

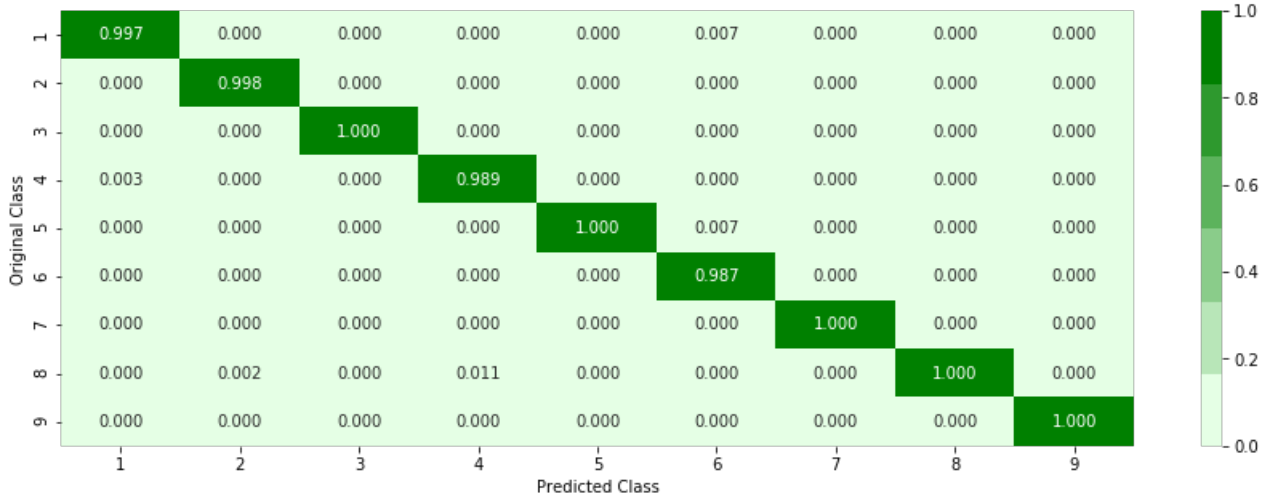




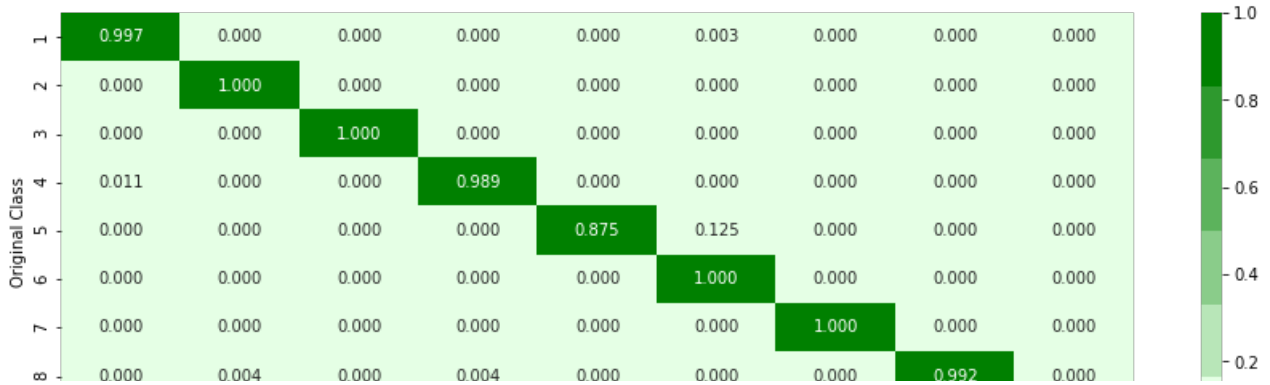

```
For values of best alpha =  3000 The train log loss is: 0.01915214143953885
For values of best alpha =  3000 The cross validation log loss is: 0.05672373738554634
For values of best alpha =  3000 The test log loss is: 0.021567213998003338
Number of misclassified points  0.22999080036798528
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```

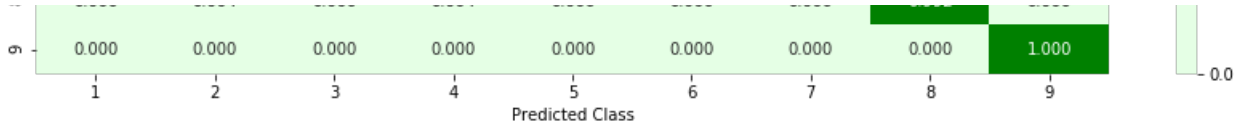



```
---------------------------------------------------- Precision matrix -----------------------------
----------------
```




```
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.]
```

4.4.5 Xgboost Classifier with best hyperparameters

In [0]:

```
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 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    7.4s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   54.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  9.0min finished
```

Out[0]:

```
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)
```

In [0]:

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

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 5, 'learning_rate': 0.15, 'colsample_bytree':
0.5}
```

In [0]:

```
# 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, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------

x_cfl=XGBClassifier(n_estimators=1000,subsample=1,learning_rate=0.15,colsample_bytree=0.5,max_depth
=5)
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 0.018748260521589 19
cv loss 0.048745261558111 57
test loss 0.020774260203484322
```

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

4.5.1. Merging both asm and byte file features

In [0]:

```
result.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...

5 rows × 260 columns

In [0]:

```
result_asm.head()
```

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.000072	...	0.000343
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000343
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000248
3	3X2nY7iQaPBlWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000114
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.000072	...	0.000229

5 rows × 53 columns

In [0]:

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

```
(10868, 260)
(10868, 53)
```

In [0]:

```
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()
```

Out[0]:

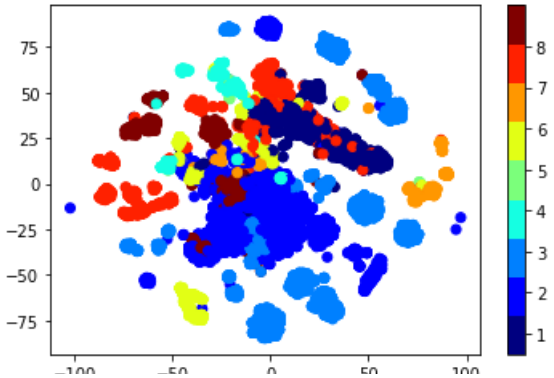
	0	1	2	3	4	5	6	7	8	9	...	:dword	edx
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.032784	0.01541
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.010846	0.00496
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.006773	0.00009
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.001028	0.00034
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.009150	0.00034

5 rows × 306 columns

4.5.2. Multivariate Analysis on final fearures

In [0]:

```
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()
```



-100 -50 0 50 100

4.5.3. Train and Test split

In [0]:

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

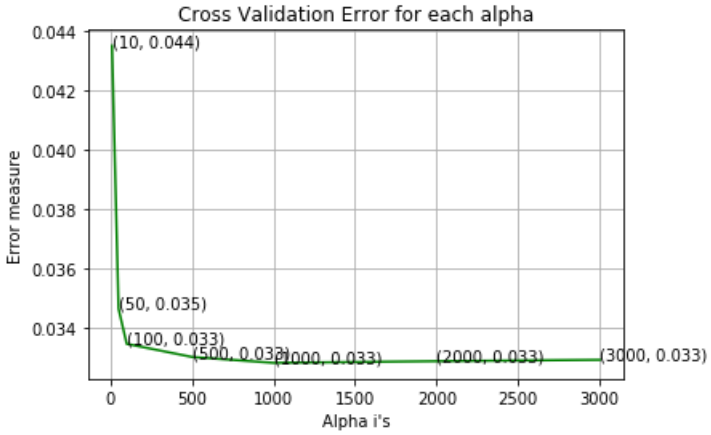
4.5.4. Random Forest Classifier on final features

In [0]:

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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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


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

predict_y = sig_clf.predict_proba(X_train_merge)
```

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

```
log_loss for c =  10 is 0.043515708652402604
log_loss for c =  50 is 0.03463883964436407
log_loss for c =  100 is 0.03344930298425949
log_loss for c =  500 is 0.03300730260797196
log_loss for c =  1000 is 0.032807861366077926
log_loss for c =  2000 is 0.032864011313455926
log_loss for c =  3000 is 0.03291032963567509
```




```
For values of best alpha =  1000 The train log loss is: 0.017368059329597313
For values of best alpha =  1000 The cross validation log loss is: 0.032807861366077926
For values of best alpha =  1000 The test log loss is: 0.02994922971777716
```

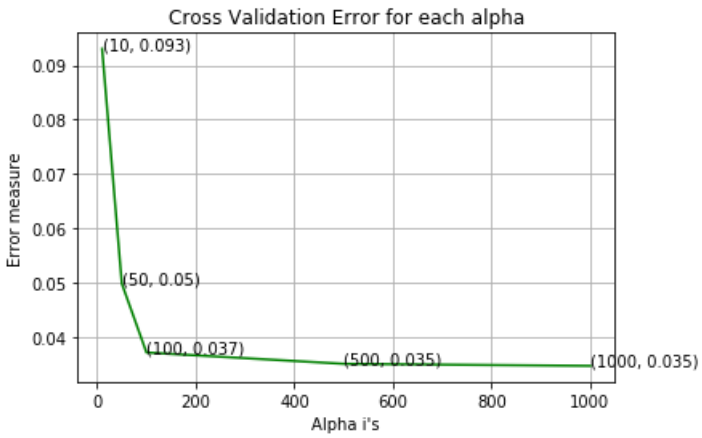
4.5.5. XgBoost Classifier on final features

In [0]:

```
# 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, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------

alpha=[10,50,100,500,1000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
```

```
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
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_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_merge, predict_y))
```

```
log_loss for c =  10 is 0.09303395536070587
log_loss for c =  50 is 0.04978639486226878
log_loss for c =  100 is 0.03709537758609293
log_loss for c =  500 is 0.03501647336229862
log_loss for c =  1000 is 0.03464059359017878
```




```
For values of best alpha =  1000 The train log loss is: 0.010899295963969774
For values of best alpha =  1000 The cross validation log loss is: 0.03464059359017878
For values of best alpha =  1000 The test log loss is: 0.029431235370045893
```

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

In [0]:

```
%%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_merge, y_train_merge)
```

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   31.7s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 17.0min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 29.6min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 35.1min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 39.1min finished
```

```
CPU times: user 4min 3s, sys: 499 ms, total: 4min 4s
Wall time: 43min 7s
```

In [0]:

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

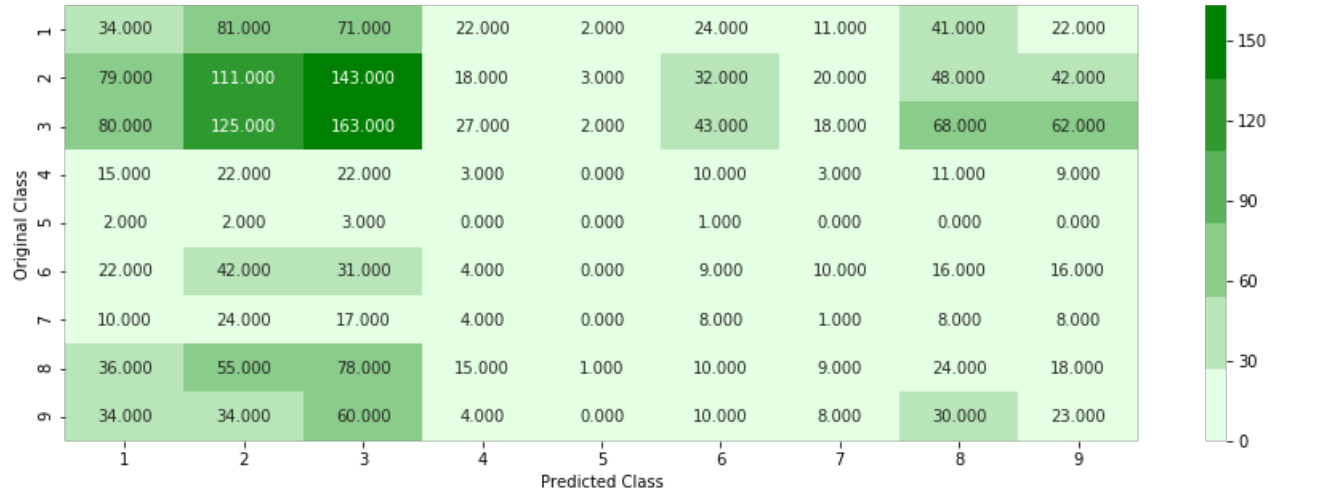
```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1
}
```

In [0]:

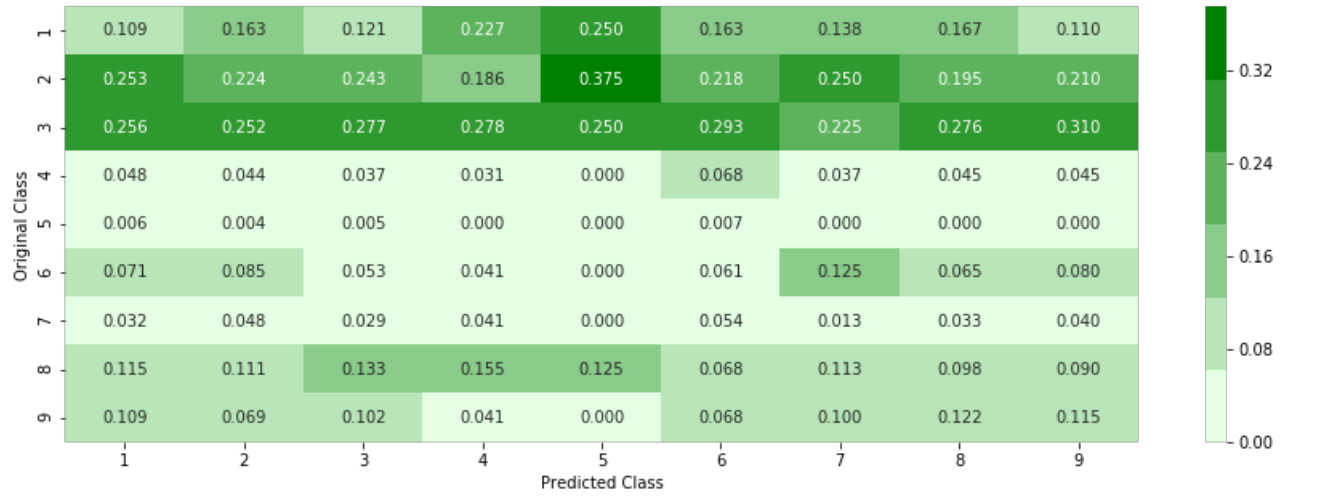
```
# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', 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, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------
%%time
x_cfl=XGBClassifier(n_estimators=500,max_depth=3,learning_rate=0.05,colsample_bytree=1,subsample=1,
ead=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  1000 The train log loss is: 0.01123131234723137
For values of best alpha =  1000 The cross validation log loss is: 0.0361269979787257
For values of best alpha =  1000 The test log loss is: 0.03037244493273
```

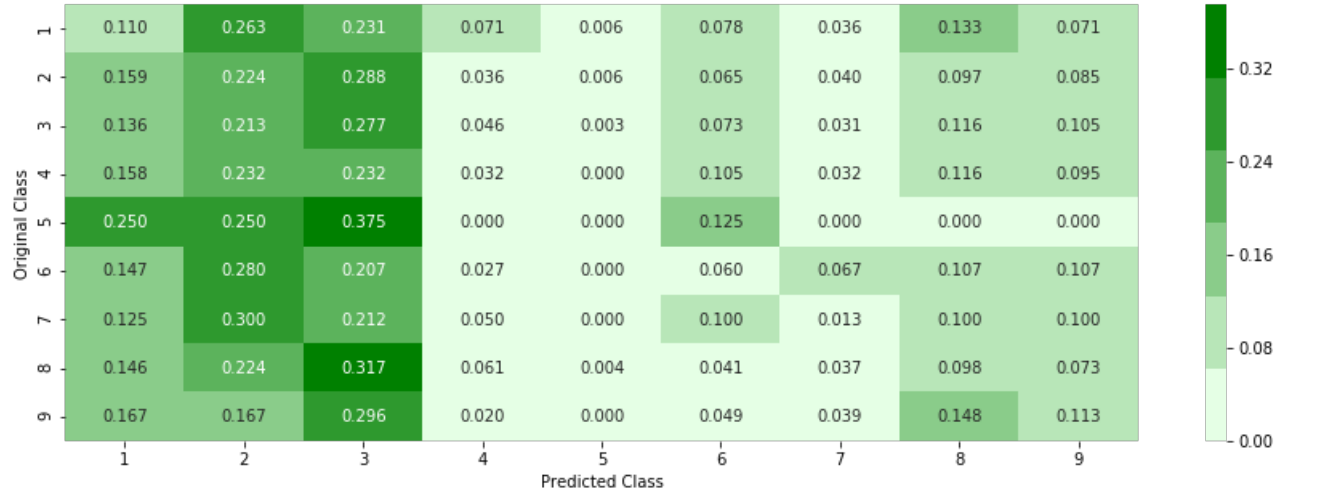
```
Number of misclassified points  83.07267709291628
---------------------------------------------------- Confusion matrix ------------------------------
----------------
```




```
---------------------------------------------------- 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.]
CPU times: user 13min 57s, sys: 707 ms, total: 13min 57s
Wall time: 13min 59s
```

Below we computing feature Engineering using Asm Files

Opcode vectorization

In [0]:

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

In [0]:

```
asm_opcode_bigram = []
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        asm_opcode_bigram.append(v + ' ' + opcodes[j])
len(asm_opcode_bigram)
```

Out[0]:

```
676
```

In [0]:

```
asm_opcode_trigram = []
for i, v in enumerate(opcodes):
    for j in range(0, len(opcodes)):
        for k in range(0, len(opcodes)):
            asm_opcode_trigram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k])
len(asm_opcode_trigram)
```

Out[0]:

```
17576
```

In [0]:

```
#intially create five folders #first #second #thrid #fourth #fifth #this code tells us about rando
m 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)
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='asm_first'
folder_2 ='asm_second'
folder_3 ='asm_third'
folder_4 ='asm_fourth'
folder_5 ='asm_fifth'
folder_6 ='asm_sixth'
folder_7 ='asm_seventh'
folder_8 ='asm_eigth'
folder_9 ='asm_ninth'
folder_10 ='asm_tenth'
for i in [folder_1,folder_2,folder_3,folder_4,folder_5,folder_6,folder_7,folder_8,folder_9,folder_1
0]:
    if not os.path.isdir(i):
        os.makedirs(i)

source='asmFiles/'
files = os.listdir('asmFiles')
#ID=df['Id'].tolist()
data=range(0,10868)
#r.shuffle(data)
count=0
for i in tqdm(range(0,10868),position=0):
    if i % 10==0:
```

```
        shutil.copy(source+files[data[i]],'asm_first')
    elif i%10==1:
        shutil.copy(source+files[data[i]],'asm_second')
    elif i%10==2:
        shutil.copy(source+files[data[i]],'asm_third')
    elif i%10==3:
        shutil.copy(source+files[data[i]],'asm_fourth')
    elif i%10==4:
        shutil.copy(source+files[data[i]],'asm_fifth')
    elif i%10==5:
        shutil.copy(source+files[data[i]],'asm_sixth')
    elif i%10==6:
        shutil.copy(source+files[data[i]],'asm_seventh')
    elif i%10==7:
        shutil.copy(source+files[data[i]],'asm_eigth')
    elif i%10==8:
        shutil.copy(source+files[data[i]],'asm_ninth')
    elif i%10==9:
        shutil.copy(source+files[data[i]],'asm_tenth')
```

```
100%|██████████| 10868/10868 [1:01:44<00:00,  2.93it/s]
```

In [0]:

```
path, dirs, files = next(os.walk("asm_first"))
file_count = len(files)
print('No of files in folder-1',file_count)

path, dirs, files = next(os.walk("asm_second"))
file_count = len(files)
print('No of files in folder-2',file_count)

path, dirs, files = next(os.walk("asm_third"))
file_count = len(files)
print('No of files in folder-3',file_count)

path, dirs, files = next(os.walk("asm_fourth"))
file_count = len(files)
print('No of files in folder-4',file_count)

path, dirs, files = next(os.walk("asm_fifth"))
file_count = len(files)
print('No of files in folder-5',file_count)

path, dirs, files = next(os.walk("asm_sixth"))
file_count = len(files)
print('No of files in folder-6',file_count)

path, dirs, files = next(os.walk("asm_seventh"))
file_count = len(files)
print('No of files in folder-7',file_count)

path, dirs, files = next(os.walk("asm_eigth"))
file_count = len(files)
print('No of files in folder-8',file_count)

path, dirs, files = next(os.walk("asm_ninth"))
file_count = len(files)
print('No of files in folder-9',file_count)

path, dirs, files = next(os.walk("asm_tenth"))
file_count = len(files)
print('No of files in folder-10',file_count)
```

```
No of files in folder-1 1087
No of files in folder-2 1087
No of files in folder-3 1087
No of files in folder-4 1087
No of files in folder-5 1087
No of files in folder-6 1087
No of files in folder-7 1087
No of files in folder-8 1087
No of files in folder-9 1086
No of files in folder-10 1086
```

Below we getting Opcode features

In [0]:

```
%%time
from tqdm import tqdm
import scipy
from sklearn.feature_extraction.text import CountVectorizer
def opcode_collect_1():
    op_file = open("opcode_file_1.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_first')):
        opcode_str = ""
        with codecs.open('asm_first/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()


def opcode_collect_2():
    op_file = open("opcode_file_2.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_second')):
        opcode_str = ""
        with codecs.open('asm_second/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()


def opcode_collect_3():
    op_file = open("opcode_file_3.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_third')):
        opcode_str = ""
        with codecs.open('asm_third/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()

def opcode_collect_4():
    op_file = open("opcode_file_4.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_fourth')):
        opcode_str = ""
        with codecs.open('asm_fourth/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()



def opcode_collect_5():
    op_file = open("opcode_file_5.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_fifth')):
        opcode_str = ""
        with codecs.open('asm_fifth/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
```

```
    op_file.close()


def opcode_collect_6():
    op_file = open("opcode_file_6.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_sixth')):
        opcode_str = ""
        with codecs.open('asm_sixth/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()


def opcode_collect_7():
    op_file = open("opcode_file_7.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_seventh')):
        opcode_str = ""
        with codecs.open('asm_seventh/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()


def opcode_collect_8():
    op_file = open("opcode_file_8.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_eigth')):
        opcode_str = ""
        with codecs.open('asm_eigth/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()


def opcode_collect_9():
    op_file = open("opcode_file_9.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_ninth')):
        opcode_str = ""
        with codecs.open('asm_ninth/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()


def opcode_collect_10():
    op_file = open("opcode_file_10.txt", "w+")
    for asmfile in tqdm(os.listdir('asm_tenth')):
        opcode_str = ""
        with codecs.open('asm_tenth/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 78.9 µs
```

In [0]:

```
%%time
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=opcode_collect_1)
    p2=Process(target=opcode_collect_2)
    p3=Process(target=opcode_collect_3)
    p4=Process(target=opcode_collect_4)
    p5=Process(target=opcode_collect_5)
    p6=Process(target=opcode_collect_6)
    p7=Process(target=opcode_collect_7)
    p8=Process(target=opcode_collect_8)
    p9=Process(target=opcode_collect_9)
    p10=Process(target=opcode_collect_10)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    p6.start()
    p7.start()
    p8.start()
    p9.start()
    p10.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()
    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()


if __name__=="__main__":
    main()
```

```
100%|██████████| 1087/1087 [1:18:59<00:00,  4.36s/it]
100%|██████████| 1086/1086 [1:21:48<00:00,  4.52s/it]
100%|██████████| 1087/1087 [1:24:00<00:00,  4.64s/it]
100%|██████████| 1087/1087 [1:24:37<00:00,  4.67s/it]
100%|██████████| 1087/1087 [1:24:44<00:00,  4.68s/it]
100%|██████████| 1087/1087 [1:25:13<00:00,  4.70s/it]
100%|██████████| 1087/1087 [1:25:40<00:00,  4.73s/it]
100%|██████████| 1087/1087 [1:25:41<00:00,  4.73s/it]
100%|██████████| 1086/1086 [1:25:42<00:00,  4.74s/it]
100%|██████████| 1087/1087 [1:25:45<00:00,  4.73s/it]
```

```
CPU times: user 11.5 s, sys: 4.28 s, total: 15.8 s
Wall time: 1h 25min 46s
```

Applying Bi and Tri Gram vectorizer on opcode Features

In [0]:

```
%%time
from tqdm import tqdm
import scipy
from sklearn.feature_extraction.text import CountVectorizer
def firstprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect1 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_1.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect1[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect1.npz', opcodebivect1)
```

```
def secondprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect2 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_2.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect2[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect2.npz', opcodebivect2)

def thirdprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect3 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_3.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect3[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect3.npz', opcodebivect3)

def fourthprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect4 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_4.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect4[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect4.npz', opcodebivect4)

def fifthprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect5 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_5.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect5[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect5.npz', opcodebivect5)

def sixthprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect6 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_6.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect6[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect6.npz', opcodebivect6)

def seventhprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect7 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_7.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect7[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect7.npz', opcodebivect7)

def eigthprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect8 = scipy.sparse.csr_matrix((1087, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_8.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodebivect8[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect8.npz', opcodebivect8)

def ninthprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect9 = scipy.sparse.csr_matrix((1086, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_9.txt').read().split('\n')
    for indx in tqdm(range(1086)):
        opcodebivect9[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect9.npz', opcodebivect9)

def tenthprocess():
    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigram)
    opcodebivect10 = scipy.sparse.csr_matrix((1086, len(asm_opcode_bigram)))
    raw_opcode = open('opcode_file_10.txt').read().split('\n')
    for indx in tqdm(range(1086)):
        opcodebivect10[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodebivect10.npz', opcodebivect10)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 141 µs
```

In [0]:

```
%%time
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)
    p6=Process(target=sixthprocess)
    p7=Process(target=seventhprocess)
    p8=Process(target=eigthprocess)
    p9=Process(target=ninthprocess)
    p10=Process(target=tenthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    p6.start()
    p7.start()
    p8.start()
    p9.start()
    p10.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()
    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()


if __name__=="__main__":
    main()
```

```
100%|██████████| 1087/1087 [00:34<00:00, 31.79it/s]
100%|██████████| 1087/1087 [00:33<00:00, 32.19it/s]
100%|██████████| 1087/1087 [00:35<00:00, 30.45it/s]
100%|██████████| 1086/1086 [00:34<00:00, 31.60it/s]
100%|██████████| 1086/1086 [00:34<00:00, 31.77it/s]
100%|██████████| 1087/1087 [00:36<00:00, 29.82it/s]
100%|██████████| 1087/1087 [00:36<00:00, 30.08it/s]
100%|██████████| 1087/1087 [00:35<00:00, 30.63it/s]
100%|██████████| 1087/1087 [00:35<00:00, 30.29it/s]
100%|██████████| 1087/1087 [00:36<00:00, 29.80it/s]
```

```
CPU times: user 1.98 s, sys: 2.88 s, total: 4.86 s
Wall time: 39.5 s
```

In [0]:

```
%%time
opcode_bigram_vect1=scipy.sparse.load_npz('opcodebivect1.npz')
opcode_bigram_vect2=scipy.sparse.load_npz('opcodebivect2.npz')
opcode_bigram_vect3=scipy.sparse.load_npz('opcodebivect3.npz')
opcode_bigram_vect4=scipy.sparse.load_npz('opcodebivect4.npz')
opcode_bigram_vect5=scipy.sparse.load_npz('opcodebivect5.npz')
opcode_bigram_vect6=scipy.sparse.load_npz('opcodebivect6.npz')
opcode_bigram_vect7=scipy.sparse.load_npz('opcodebivect7.npz')
opcode_bigram_vect8=scipy.sparse.load_npz('opcodebivect8.npz')
opcode_bigram_vect9=scipy.sparse.load_npz('opcodebivect9.npz')
opcode_bigram_vect10=scipy.sparse.load_npz('opcodebivect10.npz')
```

```
CPU times: user 104 ms, sys: 28 ms, total: 132 ms
Wall time: 155 ms
```

Wall time: 155 ms

In [0]:

```
%%time
opcode_bi_grams =
scipy.sparse.vstack((opcode_bigram_vect1,opcode_bigram_vect2,opcode_bigram_vect3,opcode_bigram_vect
code_bigram_vect5,opcode_bigram_vect6,opcode_bigram_vect7,opcode_bigram_vect8,opcode_bigram_vect9,
e_bigram_vect10)).tocsr()
```

```
CPU times: user 20 ms, sys: 12 ms, total: 32 ms
Wall time: 30 ms
```

In [0]:

```
%%time
scipy.sparse.save_npz('opcode_bi_grams.npz', opcode_bi_grams)
```

```
CPU times: user 1.06 s, sys: 52 ms, total: 1.11 s
Wall time: 1.12 s
```

In [0]:

```
%%time
import scipy
final_opcode_bigram=scipy.sparse.load_npz('opcode_bi_grams.npz')
```

```
CPU times: user 88 ms, sys: 16 ms, total: 104 ms
Wall time: 340 ms
```

In [0]:

```
final_opcode_bigram
```

Out[0]:

```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'
 with 1877309 stored elements in Compressed Sparse Row format>
```

Tri gram Vectorizer on Opcode Features

In [0]:

```
%%time
from tqdm import tqdm
import scipy
from sklearn.feature_extraction.text import CountVectorizer
def firstprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect1 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_1.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect1[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect1.npz', opcodetrivect1)

def secondprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect2 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_2.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect2[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect2.npz', opcodetrivect2)

def thirdprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect3 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_3.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect3[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect3.npz', opcodetrivect3)
```

```
def fourthprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect4 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_4.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect4[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect4.npz', opcodetrivect4)

def fifthprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect5 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_5.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect5[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect5.npz', opcodetrivect5)

def sixthprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect6 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_6.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect6[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect6.npz', opcodetrivect6)

def seventhprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect7 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_7.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect7[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect7.npz', opcodetrivect7)

def eigthprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect8 = scipy.sparse.csr_matrix((1087, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_8.txt').read().split('\n')
    for indx in tqdm(range(1087)):
        opcodetrivect8[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect8.npz', opcodetrivect8)

def ninthprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect9 = scipy.sparse.csr_matrix((1086, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_9.txt').read().split('\n')
    for indx in tqdm(range(1086)):
        opcodetrivect9[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect9.npz', opcodetrivect9)

def tenthprocess():
    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trigram)
    opcodetrivect10 = scipy.sparse.csr_matrix((1086, len(asm_opcode_trigram)))
    raw_opcode = open('opcode_file_10.txt').read().split('\n')
    for indx in tqdm(range(1086)):
        opcodetrivect10[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
    scipy.sparse.save_npz('opcodetrivect10.npz', opcodetrivect10)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 69.9 µs
```

In [0]:

```
%%time
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)
    p6=Process(target=sixthprocess)
    p7=Process(target=seventhprocess)
    p8=Process(target=eigthprocess)
```

```
    p8=Process(target=eighthprocess)
    p9=Process(target=ninthprocess)
    p10=Process(target=tenthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    p6.start()
    p7.start()
    p8.start()
    p9.start()
    p10.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()
    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()


if __name__=="__main__":
    main()
```

```
100%|██████████| 1086/1086 [02:45<00:00,  6.56it/s]
100%|██████████| 1087/1087 [02:51<00:00,  6.35it/s]
100%|██████████| 1086/1086 [02:51<00:00,  6.35it/s]
100%|██████████| 1087/1087 [02:52<00:00,  6.30it/s]
100%|██████████| 1087/1087 [02:51<00:00,  6.32it/s]
100%|██████████| 1087/1087 [02:54<00:00,  6.25it/s]
100%|██████████| 1087/1087 [02:52<00:00,  6.29it/s]
100%|██████████| 1087/1087 [02:55<00:00,  6.21it/s]
100%|██████████| 1087/1087 [02:56<00:00,  6.17it/s]
100%|██████████| 1087/1087 [02:57<00:00,  6.12it/s]
```

```
CPU times: user 6.65 s, sys: 4.24 s, total: 10.9 s
Wall time: 3min
```

In [0]:

```
%%time
opcode_trigram_vect1=scipy.sparse.load_npz('opcodetrivect1.npz')
opcode_trigram_vect2=scipy.sparse.load_npz('opcodetrivect2.npz')
opcode_trigram_vect3=scipy.sparse.load_npz('opcodetrivect3.npz')
opcode_trigram_vect4=scipy.sparse.load_npz('opcodetrivect4.npz')
opcode_trigram_vect5=scipy.sparse.load_npz('opcodetrivect5.npz')
opcode_trigram_vect6=scipy.sparse.load_npz('opcodetrivect6.npz')
opcode_trigram_vect7=scipy.sparse.load_npz('opcodetrivect7.npz')
opcode_trigram_vect8=scipy.sparse.load_npz('opcodetrivect8.npz')
opcode_trigram_vect9=scipy.sparse.load_npz('opcodetrivect9.npz')
opcode_trigram_vect10=scipy.sparse.load_npz('opcodetrivect10.npz')
```

```
CPU times: user 404 ms, sys: 28 ms, total: 432 ms
Wall time: 431 ms
```

In [0]:

```
%%time
opcode_tri_grams =
scipy.sparse.vstack((opcode_trigram_vect1,opcode_trigram_vect2,opcode_trigram_vect3,opcode_trigram_
4,opcode_trigram_vect5,opcode_trigram_vect6,opcode_trigram_vect7,opcode_trigram_vect8,opcode_trigra
ct9,opcode_trigram_vect10)).tocsr()
```

```
CPU times: user 56 ms, sys: 64 ms, total: 120 ms
Wall time: 29.8 ms
```

In [0]:

```
%%time
scipy.sparse.save_npz('opcode_tri_grams.npz', opcode_tri_grams)
```

```
CPU times: user 4.14 s, sys: 176 ms, total: 4.31 s
Wall time: 4.32 s
```

In [0]:

```
%%time
final_opcode_trigram=scipy.sparse.load_npz('opcode_tri_grams.npz')
```

```
CPU times: user 624 ms, sys: 180 ms, total: 804 ms
Wall time: 804 ms
```

In [0]:

```
final_opcode_trigram
```

Out[0]:

```
<10868x17576 sparse matrix of type '<class 'numpy.float64'>'
 with 7332672 stored elements in Compressed Sparse Row format>
```

Fetching Class labels for Opcode vectorized features we store and will use at end

In [0]:

```
#getting ids size class labels for each folder files
asm_files = os.listdir('asm_first')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_first/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_1 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_second')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_second/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_2 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_third')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
```

```
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_third/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_3 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_fourth')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_fourth/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_4 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_fifth')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_fifth/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_5 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_sixth')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_sixth/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_6 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_seventh')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
```

```
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_seventh/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_7 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_eigth')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_eigth/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_8 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_ninth')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_ninth/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_9 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})

asm_files = os.listdir('asm_tenth')
file_names = Y['Id'].tolist()
class_y = Y['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('asm_tenth/'+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 == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

asm_size_byte_10 = pd.DataFrame({'ID': fnames, 'size': sizebytes, 'Class': class_bytes})
```

In [0]:

```
asm_fnames =
[asm_size_byte_1,asm_size_byte_2,asm_size_byte_3,asm_size_byte_4,asm_size_byte_5,asm_size_byte_6,a
sm_size_byte_7,asm_size_byte_8,asm_size_byte_9,asm_size_byte_10]
```

In [0]:

```
asm_result_class = pd.concat(asm_fnames)
```

In [0]:

```
asm_result_class = pd.DataFrame(asm_result_class)
asm_result_class.head()
```

Out[0]:

	Class	ID	size
0	2	0Hlm4XgE1cQhC6BkMays	86.737548
1	2	4KOceFJiZ30X7NtS9lL1	14.207211
2	3	CnBE8f9tH12lyUeQVR74	0.261546
3	2	3Vhmj45EaPbB60rXolMw	79.769630
4	2	fdSHkFm8b2XBevTGCYM4	119.499329

In [0]:

```
final_opcode_bigram = pd.SparseDataFrame(normalize(final_opcode_bigram),columns=asm_opcode_bigram)
.fillna(0)
#final_opcode_bigram=final_opcode_bigram.to_dense()
final_opcode_bigram.head()
```

Out[0]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	jmp inc	...	movzx cmp	movzx call	movzx shl	movzx ror
0	0.000000	0.000697	0.0	0.001413	0.000000	0.002209	0.001992	0.0	0.0	0.000000	...	0.000000	0.0	0.0	0.0
1	0.002604	0.003079	0.0	0.006594	0.006897	0.002946	0.000000	0.0	0.0	0.111675	...	0.001621	0.0	0.0	0.0
2	0.000000	0.000000	0.0	0.000471	0.000000	0.000000	0.000000	0.0	0.0	0.000000	...	0.000000	0.0	0.0	0.0
3	0.000000	0.000813	0.0	0.002355	0.000000	0.002946	0.000000	0.0	0.0	0.000000	...	0.000000	0.0	0.0	0.0
4	0.000000	0.001917	0.0	0.003297	0.000000	0.002946	0.000000	0.0	0.0	0.000000	...	0.000000	0.0	0.0	0.0

5 rows × 676 columns

In [0]:

```
new_cols = np.hstack((asm_result_class.columns,final_opcode_bigram.columns))
```

In [0]:

```
final_opcode_bigrams = np.hstack((asm_result_class,final_opcode_bigram))
final_opcode_bigrams = pd.DataFrame(final_opcode_bigrams,columns=new_cols)
final_opcode_bigrams.head()
```

Out[0]:

	Class	ID	size	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp
0	2	0Hlm4XgE1cQhC6BkMays	86.7375	0	0.000697188	0	0.00141309	0	0.00220913	0.0019
1	2	4KOceFJiZ30X7NtS9lL1	14.2072	0.00260417	0.00307925	0	0.00659444	0.00689655	0.00294551	0
2	3	CnBE8f9tH12lyUeQVR74	0.261546	0	0	0	0.000471032	0	0	0
3	2	3Vhmj45EaPbB60rXolMw	79.7696	0	0.000813386	0	0.00235516	0	0.00294551	0

	Class	ID	size	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp
4	2	fdSHkFm8b2XBevTGCYM4	119.499	0	0.00191727	0	0.00329722	0	0.00294551	0

5 rows × 679 columns

In [0]:

```
final_opcode_bigrams.to_csv('final_opcode_bigrams.csv')
```

In [0]:

```
final_opcode_trigram = pd.SparseDataFrame(final_opcode_trigram,columns=asm_opcode_trigram).fillna(0)
#final_opcode_bigram=final_opcode_bigram.to_dense()
final_opcode_trigram.head()
```

Out[0]:

	jmp jmp jmp	jmp jmp mov	jmp jmp retf	jmp jmp push	jmp jmp pop	jmp jmp xor	jmp jmp retn	jmp jmp nop	jmp jmp sub	jmp jmp inc	...	movzx movzx cmp	movzx movzx call	movzx movzx shl	movzx movzx ror	movzx movzx rol	movzx movzx jnb	movzx movzx jz	movzx movzx rtn	m m
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

5 rows × 17576 columns

In [0]:

```
new_cols_2 = np.hstack((asm_result_class.columns,final_opcode_trigram.columns))
```

In [0]:

```
final_opcode_tri_grams = np.hstack((asm_result_class,final_opcode_trigram))
final_opcode_tri_grams = pd.DataFrame(final_opcode_bigrams,columns=new_cols_2).fillna(0)
final_opcode_tri_grams.head()
```

Out[0]:

	Class	ID	size	jmp jmp jmp	jmp jmp mov	jmp jmp retf	jmp jmp push	jmp jmp pop	jmp jmp xor	jmp jmp retn	...	movzx movzx cmp	movzx movzx call	movzx movzx shl	movzx movzx ror	m m
0	2	0Hlm4XgE1cQhC6BkMays	86.737548	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
1	2	4KOceFJiZ30X7NtS9lL1	14.207211	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
2	3	CnBE8f9tH12lyUeQVR74	0.261546	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
3	2	3Vhmj45EaPbB60rXolMw	79.769630	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
4	2	fdSHkFm8b2XBevTGCYM4	119.499329	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.

5 rows × 17579 columns

In [0]:

```
final_opcode_tri_grams.to_csv('final_opcode_tri_grams.csv')
```

Extracting First 800 Px Image features from ASM Files

In [0]:

```
%%time
import array
def collect_img_asm():
    for asmfile in tqdm(os.listdir("asmFiles"),position=0):
        filename = asmfile.split('.')[0]
        file = codecs.open("asmFiles/" + asmfile, 'rb')
        filelen = os.path.getsize("asmFiles/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        scipy.misc.imsave('asm_image/' + filename + '.png',reshaped)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 13.4 µs
```

In [0]:

```
%%time
import scipy
collect_img_asm()
```

```
100%|██████████| 10868/10868 [2:01:25<00:00,  1.49it/s]
```

```
CPU times: user 1h 16min 3s, sys: 6min 52s, total: 1h 22min 56s
Wall time: 2h 1min 25s
```

In [0]:

```
#shutil.rmtree('asm_image')
```

In [0]:

```
%%time
import cv2
imagefeatures = np.zeros((10868, 800))
```

```
CPU times: user 40 ms, sys: 20 ms, total: 60 ms
Wall time: 362 ms
```

In [0]:

```
%%time
for i, asmfile in tqdm(enumerate(os.listdir("asmFiles")),position=0):
    img = cv2.imread("asm_image/" + asmfile.split('.')[0] + '.png')
    img_arr = img.flatten()[:800]
    imagefeatures[i, :] += img_arr
```

```
10868it [29:56,  6.05it/s]
```

```
CPU times: user 1h 4min 52s, sys: 11min 37s, total: 1h 16min 30s
Wall time: 29min 56s
```

In [0]:

```
%%time
from sklearn import preprocessing
imagefeatures_name = []
for i in range(800):
    imagefeatures_name.append('pix' + str(i))
final_img_df = pd.DataFrame(preprocessing.normalize(imagefeatures, axis = 0), columns = imagefeatu
res_name)
```

```
CPU times: user 280 ms, sys: 112 ms, total: 392 ms
```

```
CPU times: user 280 ms, sys: 112 ms, total: 392 ms
Wall time: 108 ms
```

In [0]:

```
final_img_df['ID'] = result_asm.ID
```

In [0]:

```
final_img_df.head()
```

Out[0]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix791	pix792
0	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927	0.012927	0.012927	0.013963	...	0.003029	0.00328
1	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.010792	0.01076
2	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.010792	0.01076
3	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.010792	0.01076
4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320	0.008320	0.008320	0.007913	...	0.010792	0.01076

5 rows × 801 columns

In [0]:

```
final_img_df.shape
```

Out[0]:

```
(10868, 801)
```

In [0]:

```
final_img_df.to_csv('final_image_features.csv')
```

Important Features using Random Forest

In [0]:

```
#referd https://github.com/sai977/microsoftmalwaredetaction
%%time
from sklearn.ensemble import RandomForestClassifier
def imp_features(data, features, keep):
    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]
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 17.4 µs
```

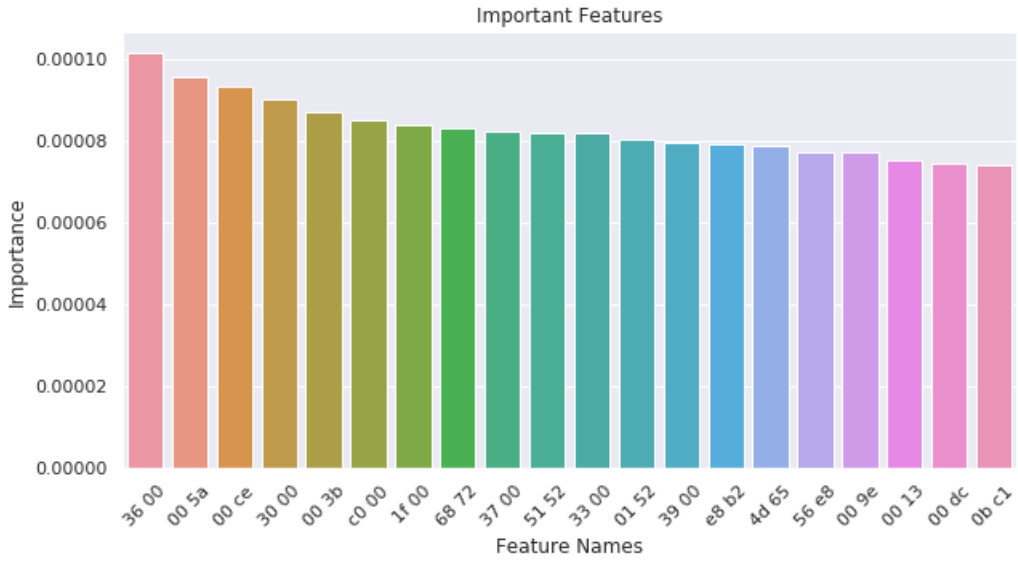
Byte features Bi-gram important features

In [0]:

```
import scipy
final_byte_bigram=scipy.sparse.load_npz('final_byte_bigram.npz')
```

In [0]:

```
byte_bi_indxes = imp_features(final_byte_bigram, byte_bigram_vocab, 1000)
```



In [0]:

```
np.save('byte_bi_indx', byte_bi_indxes)
```

In [0]:

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

In [0]:

```
top_byte_bi = np.zeros((10868, 0))
for i in tqdm(byte_bi_indxes):
    sliced = final_byte_bigram[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

```
100%|██████████| 1000/1000 [09:14<00:00,  1.80it/s]
```

In [0]:

```
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns=np.take(byte_bigram_vocab, byte_bi_indxes))
byte_bi_df.to_dense().to_csv('result_bytefile_bigrams.csv')
```

In [0]:

```
byte_bi_df = pd.read_csv('result_bytefile_bigrams.csv').drop('Unnamed: 0', axis = 1).fillna(0)
byte_bi_df.head()
```

Out[0]:

	36 00	00 5a	00 ce	30 00	00 3b	c0 00	1f 00	68 72	37 00	51 52	...	8d af	61 03
0	0.001776	0.000914	0.000836	0.002638	0.001879	0.001197	0.001403	0.000116	0.001737	0.000064	...	0.000026	0.00007
1	0.001923	0.002514	0.000887	0.002366	0.003993	0.001035	0.002958	0.001775	0.001923	0.001479	...	0.000592	0.00162
2	0.001655	0.001931	0.002207	0.002207	0.002344	0.003034	0.002758	0.002207	0.002069	0.001103	...	0.001241	0.00206
3	0.001791	0.000105	0.000105	0.001159	0.000316	0.000421	0.000211	0.000105	0.000211	0.000421	...	0.000000	0.00021

	36 00	00 5a	00 ce	30 00	00 3b	c0 00	1f 00	68 72	37 00	51 52	...	8d af	61 03
4	0.000356	0.000324	0.000130	0.001068	0.003691	0.002040	0.001619	0.000162	0.000356	0.000000	...	0.000032	0.00000

5 rows × 1000 columns

In [0]:

```
cols = np.hstack((byte_result_class.columns,byte_bi_df.columns))
```

In [0]:

```
from sklearn.preprocessing import normalize
```

In [0]:

```
byte_bi_df = np.hstack((byte_result_class,normalize(byte_bi_df)))
byte_bi_df =pd.DataFrame(byte_bi_df,columns=cols)
byte_bi_df.head()
```

Out[0]:

	Class	ID	size	36 00	00 5a	00 ce	30 00	00 3b	c0 0
0	8	gCQ70meuzrYAFaWDxZJv	2.0249	0.00178203	0.000916841	0.000839361	0.00264722	0.00188533	0.0012009
1	3	DZSwtHBVTqhivJscaoWA	8.09961	0.00234777	0.00307017	0.00108359	0.00288957	0.00487615	0.0012641
2	3	1u3qTGiRvckQZW7dBY58	8.09961	0.00196492	0.00229241	0.0026199	0.0026199	0.00278364	0.0036023
3	6	HD3SgIFw48AUEILe57oi	0.552246	0.00191837	0.000112845	0.000112845	0.0012413	0.000338535	0.0004513
4	1	bW4NY5lZng7LJeDjpQSK	0.991211	0.000360128	0.000327389	0.000130956	0.00108038	0.00373224	0.0020625

5 rows × 1003 columns

In [0]:

```
byte_bi_df.to_csv('Final_byte_bi.csv')
```

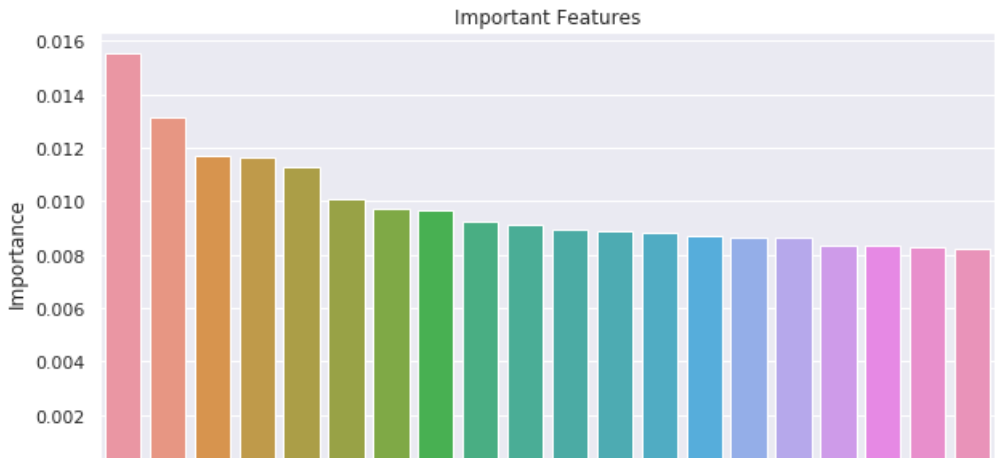
Opcode Bi-gram important features

In [0]:

```
final_opcode_bigram=scipy.sparse.load_npz('opcode_bi_grams.npz')
```

In [0]:

```
op_bi_indxes = imp_features(normalize(final_opcode_bigram, axis = 0), asm_opcode_bigram, 1000)
```



In [0]:

```
op_bi_df = pd.SparseDataFrame(normalize(final_opcode_bigram, axis = 0), columns = asm_opcode_bigram
)
for col in op_bi_df.columns:
    if col not in np.take(asm_opcode_bigram, op_bi_indxes):
        op_bi_df.drop(col, axis = 1, inplace = True)
```

In [0]:

```
op_bi_df.to_dense().to_csv('op_bi_filtered.csv')
```

In [0]:

```
op_bi_df = pd.read_csv('op_bi_filtered.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [0]:

```
op_code_bigrams = pd.read_csv('final_opcode_bigrams.csv')
```

In [0]:

```
op_bi_df['ID'] = op_code_bigrams['ID']
op_bi_df['size'] = op_code_bigrams['size']
op_bi_df.head()
```

Out[0]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	jmp inc	...	movzx shl	movzx ror	movzx rol	movzx jnb	movzx jz	movzx rtn	movzx lea	movzx movzx	
0	0.0	12.0	0.0	3.0	0.0	3.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	8.0	53.0	0.0	14.0	5.0	4.0	0.0	0.0	0.0	22.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	4
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	C
3	0.0	14.0	0.0	5.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3
4	0.0	33.0	0.0	7.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	fc

5 rows × 678 columns

In [0]:

```
op_bi_df.to_csv('Final_opcode_bi.csv')
```

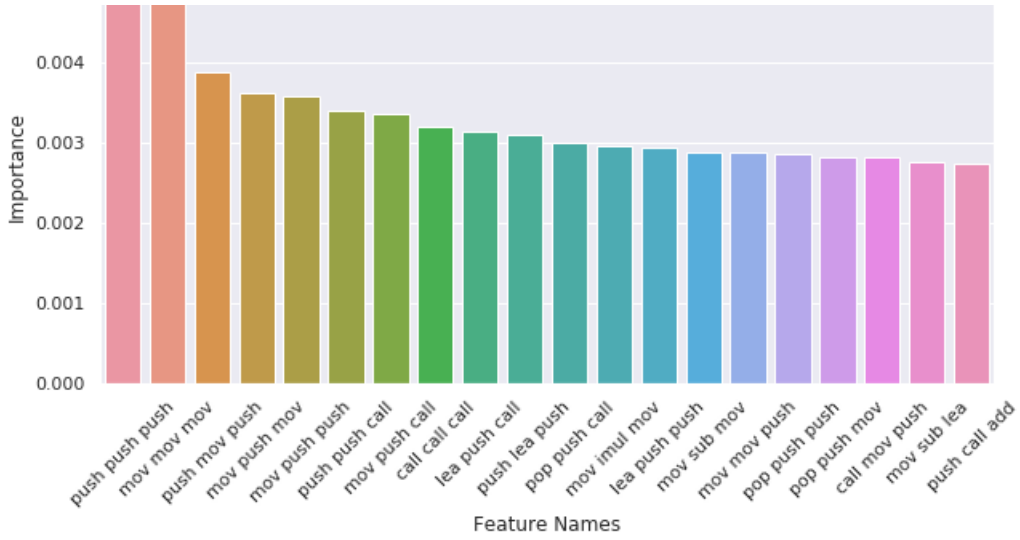
Opcode Tri gram Important features

In [0]:

```
final_opcode_trigram=scipy.sparse.load_npz('opcode_tri_grams.npz')
```

In [0]:

```
op_tri_indxes = imp_features(normalize(final_opcode_trigram, axis = 0), asm_opcode_trigram, 1000)
```



In [0]:

```
op_tri_df = pd.SparseDataFrame(normalize(final_opcode_trigram, axis = 0), columns =
asm_opcode_trigram)
op_tri_df = op_tri_df.loc[:, np.intersect1d(op_tri_df.columns, np.take(asm_opcode_trigram,
op_tri_indxes))]
```

In [0]:

```
op_tri_df.to_dense().to_csv('op_tri_filtered.csv')
```

In [0]:

```
op_tri_df = pd.read_csv('op_tri_filtered.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

In [0]:

```
op_code_trigrams = pd.read_csv('final_opcode_tri_grams.csv')
```

In [0]:

```
op_tri_df['ID'] = op_code_trigrams['ID']
op_tri_df['size'] = op_code_trigrams['size']
op_tri_df.head()
```

Out[0]:

	add add add	add add cmp	add add jmp	add add lea	add add mov	add add pop	add add push	add add sub	add add xor	add call add	...	xor sub cmp	xor sub mov	xor sub push	xor xor add	xor c
0	0.000100	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0	...	0.0	0.001694	0.0	0.0	0.0039
1	0.039824	0.005095	0.005045	0.0	0.001712	0.001891	0.003268	0.000662	0.0	0.0	...	0.0	0.001694	0.0	0.0	0.0039
2	0.000100	0.000000	0.002522	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0000
3	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.001634	0.000000	0.0	0.0	...	0.0	0.001694	0.0	0.0	0.0039
4	0.000000	0.000000	0.000000	0.0	0.000285	0.000000	0.001634	0.000000	0.0	0.0	...	0.0	0.003388	0.0	0.0	0.0039

5 rows × 1002 columns

In [0]:

```
op_tri_df.to_csv('Final_opcode_tri.csv')
```

Getting overall features here

Getting our all features here

In [0]:

```
byte_uni = pd.read_csv('result_with_size.csv')
byte_bi = pd.read_csv('Final_byte_bi.csv')
asm_uni = pd.read_csv('asmoutputfile.csv')
asm_opcode_bi = pd.read_csv('Final_opcode_bi.csv')
asm_opcode_tri = pd.read_csv('Final_opcode_tri.csv')
asm_image_fea = pd.read_csv('final_image_features.csv')
```

In [0]:

```
#cecking the common columns and we drop it
#asm_uni.columns
```

We merging all features based on id and we apply model on all the important features

In [0]:

```
byte_uni = byte_uni.drop(['Class'],axis=1)
byte_bi = byte_bi.drop(['Class','size'],axis=1)
asm_opcode_bi = asm_opcode_bi.drop(['size'],axis=1)
asm_opcode_tri = asm_opcode_tri.drop(['size'],axis=1)
```

In [0]:

```
#merging byte uni and bi
byte_uni_bi = pd.merge(byte_uni,byte_bi,on='ID',how='left')
byte_uni_bi.head()
```

Out[0]:

	Unnamed: 0_x	ID	0	1	2	3	4	5	6	7	...	8d af	61 03	e8 18
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...	0.000033	0.000018	0.000040
1	1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	...	0.000000	0.000022	0.000067
2	2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	...	0.000557	0.000278	0.001902
3	3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	...	0.000198	0.000099	0.000793
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	...	0.000130	0.000000	0.000196

5 rows × 1261 columns

In [0]:

```
byte_uni_bi = byte_uni_bi.drop(['Unnamed: 0_x'],axis=1)
byte_uni_bi.shape
```

Out[0]:

```
(10868, 1260)
```

In [0]:

```
#opcode bi opcode tri
asm_bi_tri = pd.merge(asm_opcode_bi,asm_opcode_tri,on='ID',how='left')
asm_bi_tri.head()
```

Out[0]:

	Unnamed: 0_x	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	...	xor retn push	xor retn xor	xor sub cmp	xor sub mov	xor sub push	xor xor add	xor xor cmp	
0	0	0.0	12.0	0.0	3.0	0.0	3.0	1.0	0.0	0.0	...	0.001913	0.005163	0.0	0.001694	0.0	0.0	0.003984	0

	Unnamed: 0_x	jmp	mov	retf	push	pop	xor	retn	nop	sub	...	xor retn push	xor retn xor	xor sub cmp	xor sub mov	xor sub push	xor xor add	xor xor cmp	
1	1	8.0	33.0	0.0	14.0	8.0	4.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0	0.001694	0.0	0.0	0.003984	0.
2	2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.
3	3	0.0	14.0	0.0	5.0	0.0	4.0	0.0	0.0	0.0	...	0.001913	0.000000	0.0	0.001694	0.0	0.0	0.003984	0.
4	4	0.0	33.0	0.0	7.0	0.0	4.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0	0.003388	0.0	0.0	0.003984	0.

5 rows × 1679 columns

In [0]:

```
asm_bi_tri = asm_bi_tri.drop(['Unnamed: 0_x'],axis=1)
asm_bi_tri.shape
```

Out[0]:

```
(10868, 1678)
```

In [0]:

```
#asm uni + asm opcode bi + asm  opcode tri
asm_uni_bi_tri = pd.merge(asm_uni,asm_bi_tri,on='ID')
asm_uni_bi_tri.head()
```

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	xor retn push	xor retn xor	xor sub cmp	xor sub mov	
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	0.0	0.0	0.0	0.0	0
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	0.0	0.0	0.0	0.0	0
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	0.0	0.0	0.0	0.0	0
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	0.0	0.0	0.0	0.0	0
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	0.0	0.0	0.0	0.0	0

5 rows × 1729 columns

In [0]:

```
#byte uni,bi + asm uni, bi ,tri
asm_byte_files = pd.merge(byte_uni_bi,asm_uni_bi_tri,on='ID')
asm_byte_files.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	xor retn push	xor retn xor	xor sub cmp	xor sub mov
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	0.000956	0.0	0.004508	0.018631
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	0.000956	0.0	0.000000	0.003388
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	0.003825	0.0	0.000000	0.000000
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	0.000000	0.0	0.000000	0.000000
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	0.000000	0.0	0.000000	0.000000

5 rows × 2988 columns

In [0]:

```
#asm_byte_files = asm_byte_files.drop(['Unnamed: 0_x'],axis=1)
asm_byte_files.shape
```

Out[0]:

```
(10868, 2988)
```

In [0]:

```
#asm uni bi tri byte uni bi and asm image
final_features = pd.merge(asm_byte_files,asm_image_fea,on='ID')
final_features.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	pix790	pix791	pix792	pi
0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	0.010792	0.010792	0.010768	0.01
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	0.003029	0.003029	0.003282	0.00
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	0.010792	0.010792	0.010768	0.01
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	0.003029	0.003029	0.003282	0.00
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	0.003029	0.003029	0.003282	0.00

5 rows × 3789 columns

In [0]:

```
#final_features=final_features.drop(['Unnamed: 0_x_x'],axis=1)
final_features.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	pix790	pix791	pix792	pi
0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	0.010792	0.010792	0.010768	0.01
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	0.003029	0.003029	0.003282	0.00
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	0.010792	0.010792	0.010768	0.01
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	0.003029	0.003029	0.003282	0.00
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	0.003029	0.003029	0.003282	0.00

5 rows × 3789 columns

In [0]:

```
#in final features we need to remove to this features
#Some unamed columsn noticed in features so  we removing them
#'Unnamed: 0_y_x'
#'Unnamed: 0_x_y'
#'Unnamed: 0_y_y'
#'Unnamed: 0'
final_features=final_features.drop(['Unnamed: 0_y_x','Unnamed: 0_y_y','Unnamed: 0'],axis=1)
final_features.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	pix790	pix791	pix792	pi
0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	0.010792	0.010792	0.010768	0.01
1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	0.003029	0.003029	0.003282	0.00
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	0.010792	0.010792	0.010768	0.01
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	0.003029	0.003029	0.003282	0.00
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	0.003029	0.003029	0.003282	0.00

5 rows × 3786 columns

In [0]:

```
#Some unamed columsn noticed in features so  we removing them
#Unnamed: 0_y_x
#Unnamed: 0_y_y
#Unnamed: 0
```

In [0]:

```
final_features = final_features.rename(columns = {'ID':'Id'})
final_features = pd.merge(final_features,Y,on='Id')
final_features.head()
```

Out[0]:

	Id	0	1	2	3	4	5	6	7	8	...	pix791	pix792	pix793	pi
0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	0.010792	0.010768	0.010768	0.01
1	01lsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	0.003029	0.003282	0.003282	0.00
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	0.010792	0.010768	0.010768	0.01
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	0.003029	0.003282	0.003282	0.00
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	0.003029	0.003282	0.003282	0.00

5 rows × 3787 columns

In [0]:

```
final_features.to_csv('Final_features.csv')
```

In [0]:

```
final_features = pd.read_csv('Final_features.csv',index_col=0)
```

In [0]:

```
final_features.to_pickle('final_features.pickle')
```

In [0]:

```
import pickle
files = open('/content/drive/My Drive/final_features.pickle','rb')
final_features = pickle.load(files)
```

In [10]:

```
print(final_features.shape)
final_features.head()
```

```
(10868, 3787)
```

Out[10]:

	Id	0	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e	
0	01azqd4lnC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	3205	3211	3546	4038	4096	3218	3
1	01lsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	358	340	6649	8660	447	218	6
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	2457	2655	2669	9113	2584	2788	2
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	473	516	445	808	432	403	7
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	223	237	226	406	643	213	2

	Id	0	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e

5 rows × 3787 columns

Above is our final data contains Byte Files Uni grams + Byte files Bi grams + Asm uni + Asm opcoed bi + Asm opcode tri + Asm image features

In [0]:

```
x = final_features.drop(['Id','Class'],axis=1)
y = final_features['Class']
```

In [0]:

```
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.20)
#X_train_data, X_cv_data, y_train_data, y_cv_data = train_test_split(X_train,
y_train,test_size=0.20)
```

In [8]:

```
print('Train_data :',X_train.shape,y_train.shape)
#print('Cv data :',X_cv_data.shape,y_cv_data.shape)
print('Test data :',X_test.shape,y_test.shape)
```

```
Train_data : (8694, 3785) (8694,)
Test data : (2174, 3785) (2174,)
```

In [0]:

```
#Appling Xgboost on all the features
#we limited to only xgboost because we already seen in previous models xgboost performs better and
we continue with xgboost using random search
x_cfl=XGBClassifier()
prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,cv=5)
random_cfl.fit(X_train, y_train)
print('Best Estimator :',random_cfl.best_estimator_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed: 30.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 54.1min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 76.4min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 96.3min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 107.4min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 157.6min remaining: 34.6min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 189.6min remaining: 12.1min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 197.1min finished

Best Estimator : XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0,
              learning_rate=0.05, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=1000, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.5, verbosity=1)
```

In [0]:

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

```
{'subsample': 0.5, 'n_estimators': 1000, 'max_depth': 5, 'learning_rate': 0.05,
```

'colsample_bytree': 0.5}

In [9]:

```
%%time
x_cfl=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0,
              learning_rate=0.05, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=1000, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.5, verbosity=1)
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_asm)
#print('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.00849114316562768
test loss 0.016482001009012103
CPU times: user 2h 33min 9s, sys: 6.82 s, total: 2h 33min 15s
Wall time: 2h 33min 37s
```

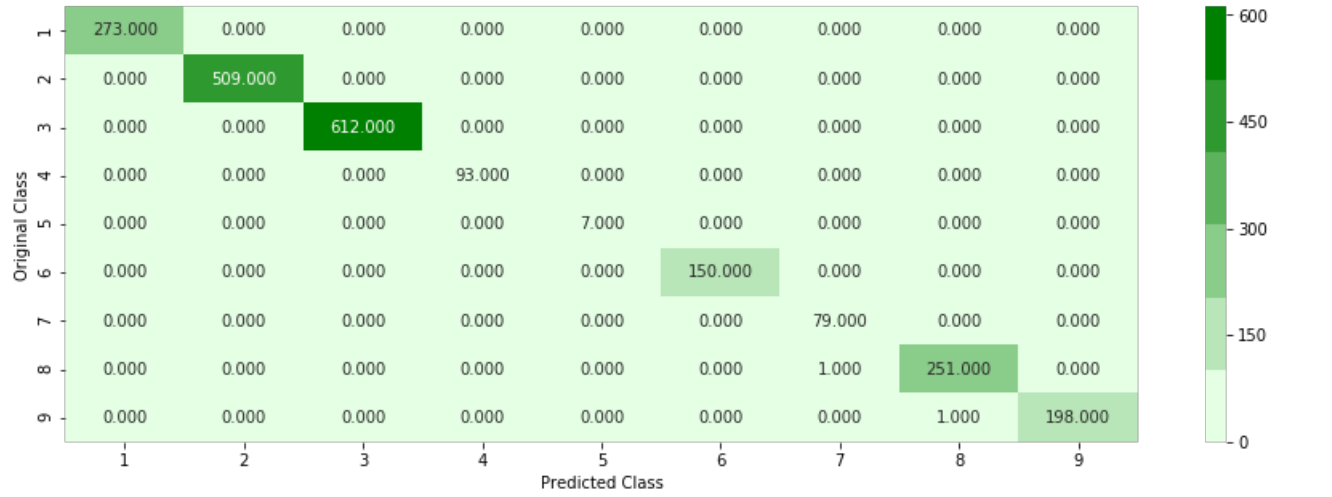
In [0]:

```
# save the model to disk
filename = '/content/drive/My Drive/c_cfl.sav'
pickle.dump(c_cfl, open(filename, 'wb'))
```

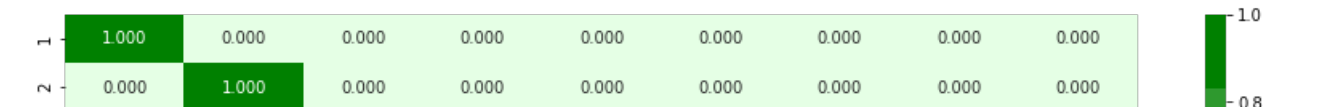
In [14]:

```
# load the model from disk
%matplotlib inline
c_cfl = pickle.load(open('/content/drive/My Drive/c_cfl.sav', 'rb'))
plot_confusion_matrix(y_test,c_cfl.predict(X_test))
```

```
Number of misclassified points  0.09199632014719411
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

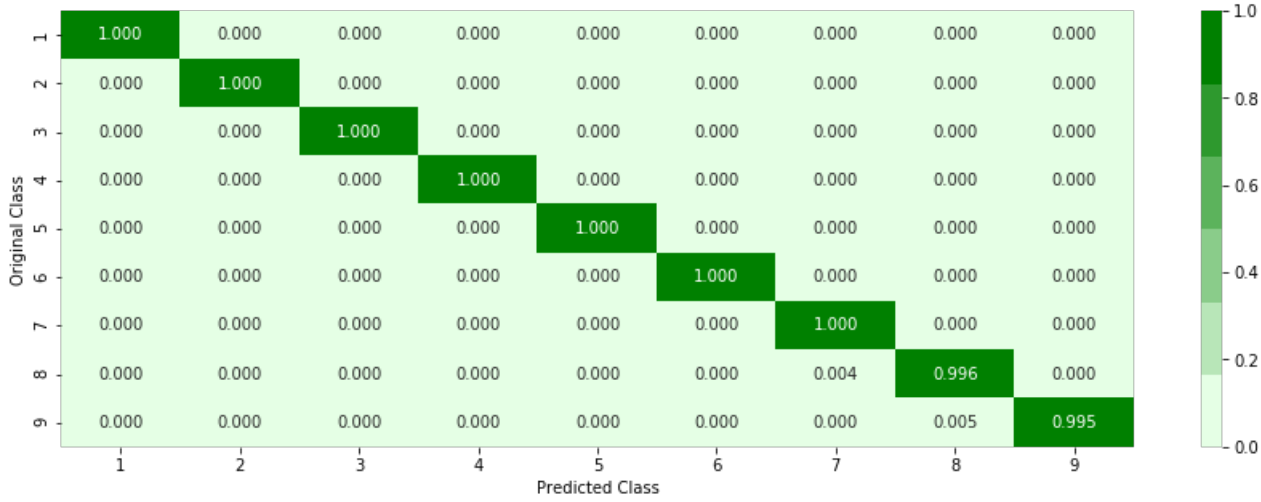



```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```



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.]

In [15]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = [ "Model","files_&_vectorizer", "Train_loss", "Test_loss"]
x.add_row(["XGB_Classifier","Byte_UNi",0.02,0.06])
x.add_row(["XGB_Classifier","Asm_uni",0.01,0.02])
x.add_row(["XGB_Classifier","Asm_uni + Byte_UNi ",0.01,0.03])
x.add_row(["XGB_Classifier","Byte_uni + byte_bi + asm_uni + asm_opcode_bi + asm_opcode_tri + asm_i
mage ",0.008,0.016])

print(x)
```

```
+----------------+-------------------------------------------------------------------------------+----
----+-----------+
|     Model      |                              files_&_vectorizer                               |
Train_loss | Test_loss |
+----------------+-------------------------------------------------------------------------------+----
----+-----------+
| XGB_Classifier |                                   Byte_UNi                                    |
0.02    |    0.06   |
| XGB_Classifier |                                   Asm_uni                                     |
0.01    |    0.02   |
| XGB_Classifier |                                Asm_uni + Byte_UNi                             |
01    |    0.03   |
| XGB_Classifier | Byte_uni + byte_bi + asm_uni + asm_opcode_bi + asm_opcode_tri + asm_image  |
0.008    |   0.016   |
+----------------+-------------------------------------------------------------------------------+----
----+-----------+
```

Observations

1. **Applied Multi-processing technique for larger files for computing Bi-grams and Asm opcode bi and tri grams**
2. **Applied Feature Engineering with reference of kaggle winner solutions and Dchad account**
3. **After combing all the some important features by using xgboost classifier we got train loss as 0.008 and lowest test log loss as 0.016**