Microsoft Malware detection Assignment

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=VLQTRILGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

```
# importing Library


import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
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
import re

from sklearn.feature_extraction.text import CountVectorizer
import scipy
```

5. Assignments

1. Add bi-grams and n-gram features on byte files and improve the log-loss
2. Using the 'dchad' github account (https://github.com/dchad/malware-detection), decrease the logloss to <=0.01
3. Watch the video (https://www.youtube.com/watch?v=VLQTRILGz5Y) that was in reference section and implement the image features to improve the logloss

5.1 Bigram BOW

```
# Bigram BOW --> Creating all the possible bigram feature for hexadecimal
bi_gram=["ID","????","0"]
for i in range(1,256*256):
    bi_gram.append(str(hex(i)).strip("0x"))
bigram=open('bigram.csv','w+')
for i in bi_gram:
    bigram.write(str(i)+",")
bigram.write("\n")

# Manual BOW bigram code
files=os.listdir("byteFiles/")
feature_matrix=np.zeros((len(files),256*256+1),dtype=np.float)
filenames2=[]
k=0

for file in files:
    filenames2.append(file)
    if(file.endswith("txt")):
        bigram.write("\n"+file+",")

    with open("byteFiles/"+file,"r") as fp:
        bigram_text=[]
        for line in fp:
            for pattern in  re.findall(r"(..\s..)+",line):
                pattern=pattern.replace(" ","")
                bigram_text.append(pattern)

        for hex_code in bigram_text:
            if hex_code=="????":
                feature_matrix[k][0] +=1

            elif '?' not in hex_code:
                feature_matrix[k][int(hex_code,16)+1] +=1

        for i in feature_matrix[k]:
            bigram.write(str(i)+",")

        k += 1
        bigram.write("\n")
bigram.close()
```

```
bigram_features=pd.read_csv('bigram.csv')
bigram_features=bigram_features.iloc[:,:-1]
```

```
# we normalize the data each column
result_bigram = normalize(bigram_features)
```

```
result_bigram.head()
```

	ID	????	0	1	2	3	4	5
0	01azqd4InC7m9JpocGv5.txt	0.000129	0.230535	0.012146	0.010006	0.013938	0.008828	0.015486
1	01IsoiSMh5gxyDYTl4CB.txt	0.000606	0.009218	0.041190	0.001032	0.001853	0.003406	0.000538
2	01jsnpXSAlgw6aPeDxrU.txt	0.000033	0.009290	0.035909	0.000318	0.000403	0.026276	0.043230
3	01kcPWA9K2BOxQeS5Rju.txt	0.000984	0.004490	0.006601	0.002621	0.002578	0.002711	0.002581
4	01SuzwMJEIXsK7A8dQbl.txt	0.000636	0.007179	0.001320	0.001191	0.004189	0.000278	0.000645

5 rows × 65538 columns

```
#storing in pickle file after normalising bigram byte file
"""result_bigram.to_pickle("result_bigram.pkl")"""
result_bigram=pd.read_pickle("result_bigram.pkl")
```

5.1.1 Modeling using Bigram byte feature only

Train Test split

```
# cleaning '.txt'  from the end of 'Id' names
ID_bigram = result_bigram.ID.apply(lambda x : x.strip(".txt"))

result_bigram.ID = ID_bigram
result_bigram.head()
```

	ID	????	0	1	2	3	4	5
0	01azqd4InC7m9JpocGv5	0.000129	0.230535	0.012146	0.010006	0.013938	0.008828	0.015486
1	01IsoiSMh5gxyDYTl4CB	0.000606	0.009218	0.041190	0.001032	0.001853	0.003406	0.000538
2	01jsnpXSAlgw6aPeDxrU	0.000033	0.009290	0.035909	0.000318	0.000403	0.026276	0.043230
3	01kcPWA9K2BOxQeS5Rju	0.000984	0.004490	0.006601	0.002621	0.002578	0.002711	0.002581
4	01SuzwMJEIXsK7A8dQbl	0.000636	0.007179	0.001320	0.001191	0.004189	0.000278	0.000645

5 rows × 65538 columns

```
# merging with class label on 'ID'
Y=pd.read_csv("trainLabels.csv")
result_bigram= result_bigram.merge(right=Y ,how='inner',left_on='ID',right_on=
'Id')

# class label
data_y = result_bigram['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_bigram.drop(['ID','C
lass','Id'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same dist
ribution 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_tra
in,test_size=0.20)
```

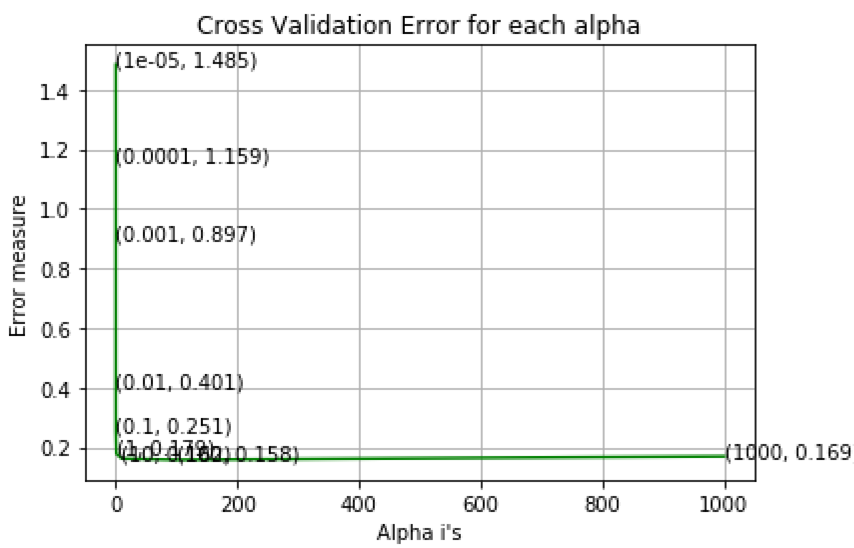
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
Number of data points in train data: 6724
Number of data points in test data: 2102
Number of data points in cross validation data: 1682
```

5.1.2 Logistic Regression

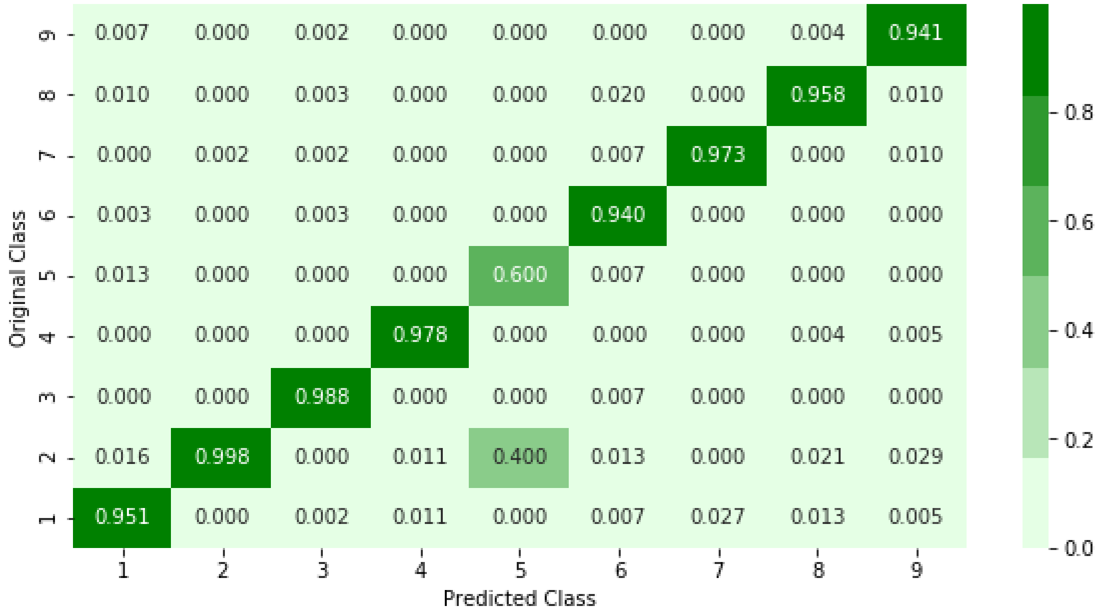
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced',n_job
s=-2)
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classe
s_, 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='bal
anced',n_jobs=-2)
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.classe
s_, 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.cl
asses_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.48467640110343
log_loss for c =  0.0001 is 1.1594554849576302
log_loss for c =  0.001 is 0.8973120541763175
log_loss for c =  0.01 is 0.4005375859385772
log_loss for c =  0.1 is 0.2511465369246042
log_loss for c =  1 is 0.1785054410142128
log_loss for c =  10 is 0.16204244832347642
log_loss for c =  100 is 0.15765745306412124
log_loss for c =  1000 is 0.1694312506386744
```

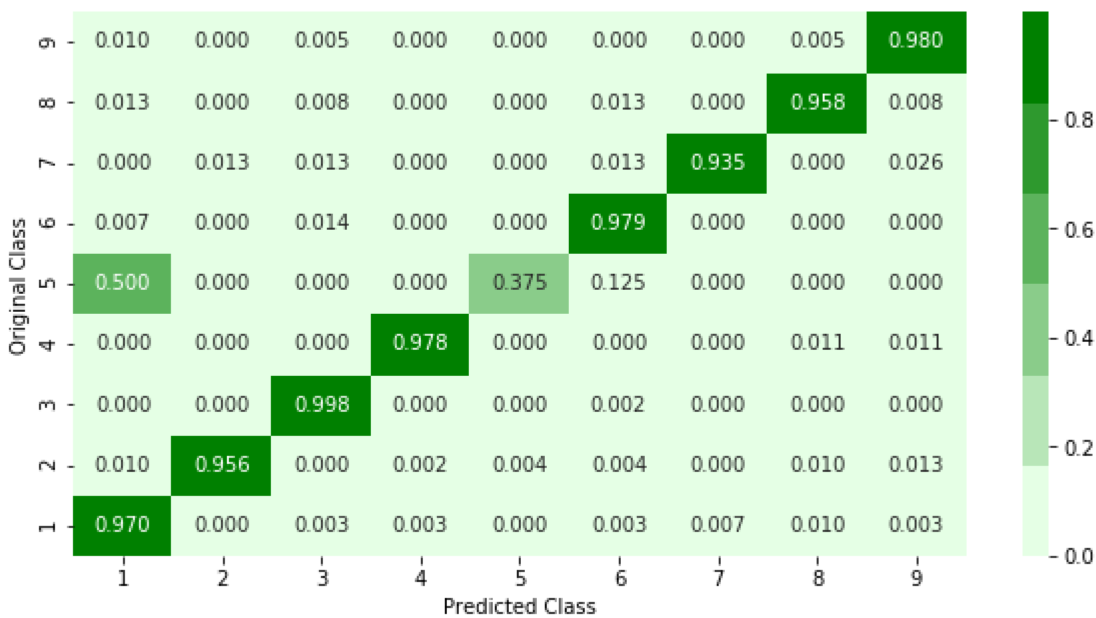



```
log loss for train data 0.062301637702642625
log loss for cv data 0.15765745306412124
log loss for test data 0.18327204878263503
Number of misclassified points  2.8544243577545196
-------------------------------------------------- Confusion matrix ------------------------------------------------
```


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




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



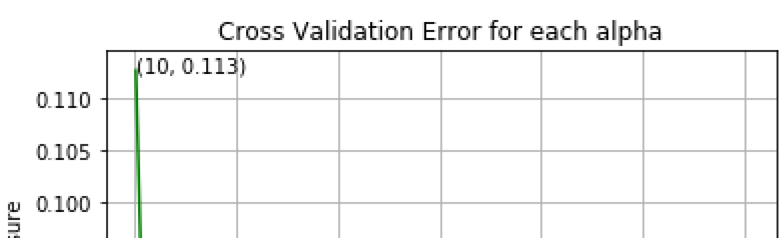

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

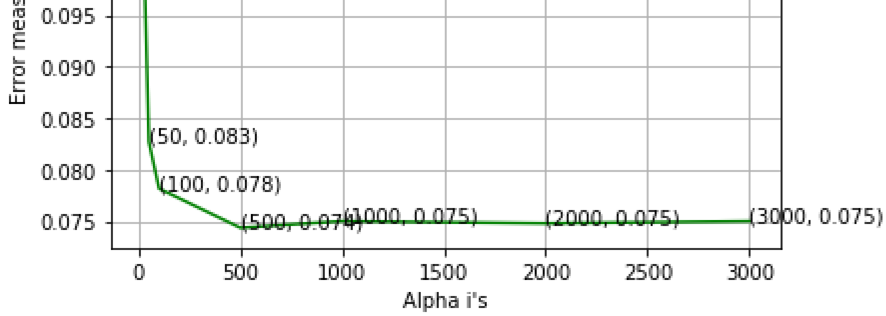
5.1.3 Random Forest Classifier

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

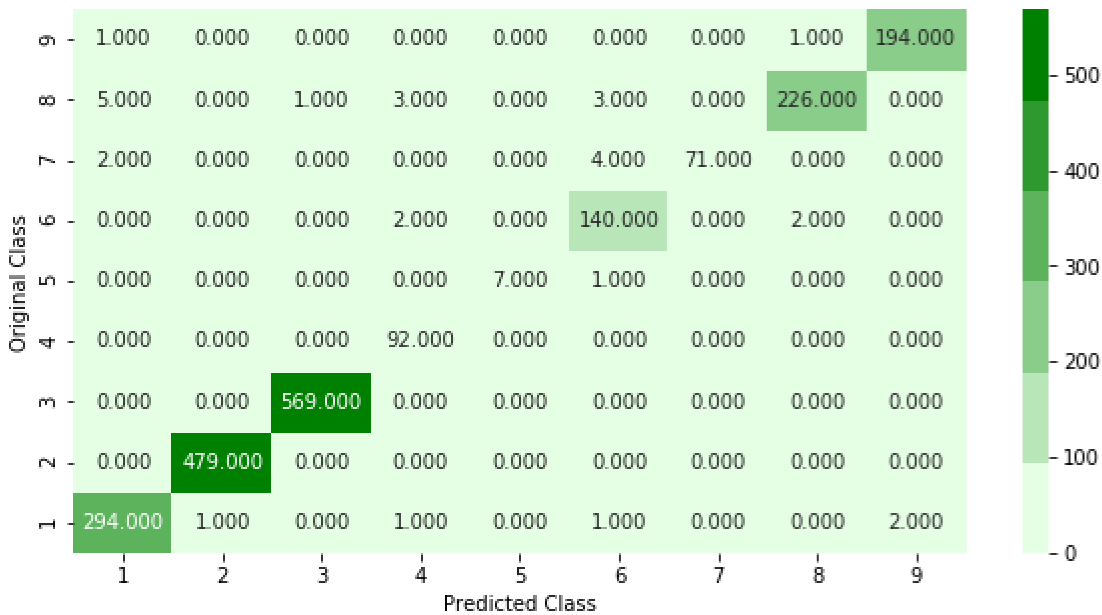
```
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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


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

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

```
log_loss for c =  10 is 0.11274641480113538
log_loss for c =  50 is 0.08270004094873512
log_loss for c =  100 is 0.07816814666224625
log_loss for c =  500 is 0.07435337316319741
log_loss for c =  1000 is 0.07500563445409537
log_loss for c =  2000 is 0.07478774985615255
log_loss for c =  3000 is 0.07500400900958992
```

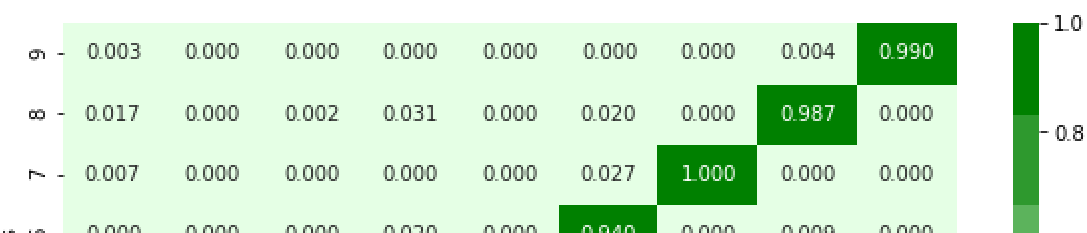




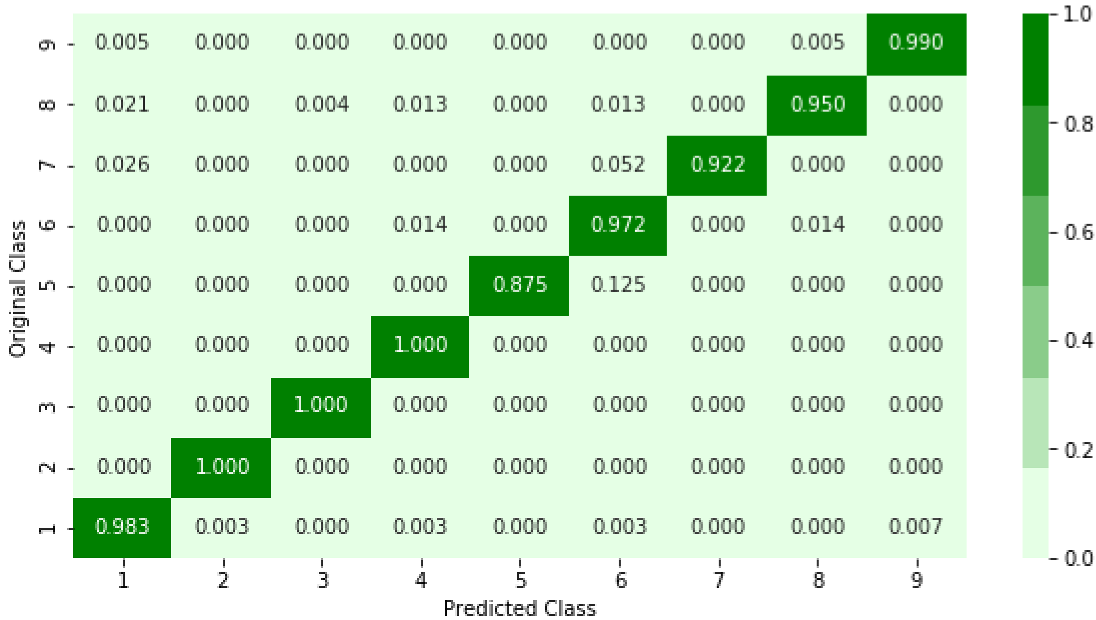

```
For values of best alpha =  500 The train log loss is: 0.022033998148102092
For values of best alpha =  500 The cross validation log loss is: 0.07435337316319741
For values of best alpha =  500 The test log loss is: 0.06883578221971019
Number of misclassified points  1.4272121788772598
------------------------------------------------ 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.]
```

! Traning Xgboost on very high dimention dta is computationaly very expensive so I am skiping running XGboost on Comple ByteBigram Featue

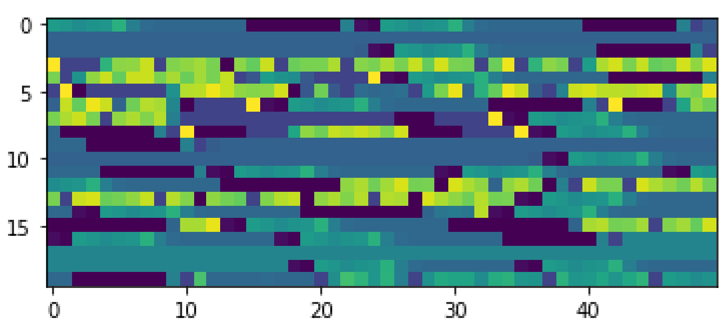
5.2 Assignment task 2 : Incorporating Image feature

```
# converting .asm file to byte file and use first 1000 bytes of each asm file a
s image filefiles=os.listdir("asmFiles/")

Y=pd.read_csv("trainLabels.csv")
```

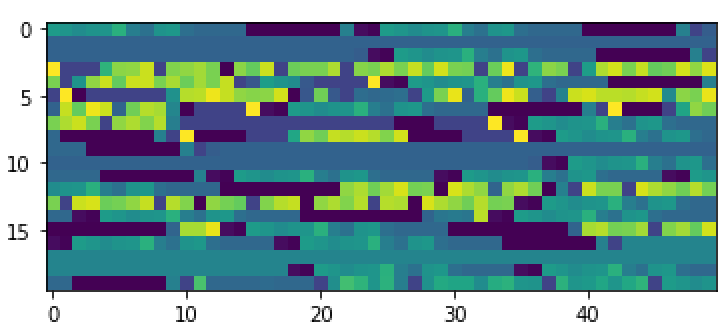
```
filenames=Y['Id'].tolist()

filenames2=[]
image_feat=[]

number_of_feature = 1000
asm_file_name= os.listdir("asmFiles/")

for file in filenames:
    filenames2.append(file)
    if(file+".asm" in asm_file_name):
        with open("asmFiles/"+file+".asm", mode='rb') as f: # b is important ->
binary
            fileContent = f.read()

            top_1000_img_feat=(fileContent[:number_of_feature])

            image_feat.append(np.frombuffer(top_1000_img_feat, dtype = np.uint8
))
```

```
for i in range(5):
    print(f"example of asm image : {i+1}")
    plt.imshow(image_feat_asm[i].reshape(20,50))
    plt.show()
```

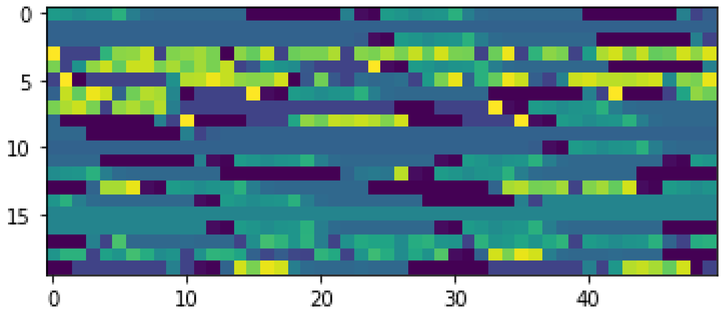
```
example of asm image : 1
```



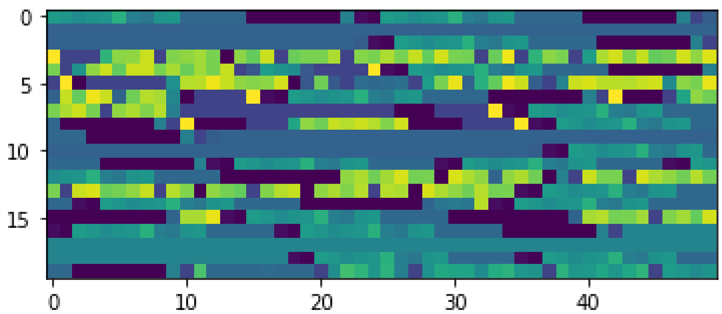
```
example of asm image : 2
```



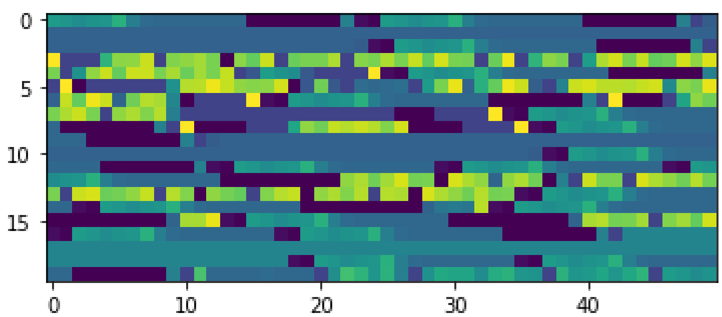
```
example of asm image : 3
```




```
example of asm image : 4
```




```
example of asm image : 5
```

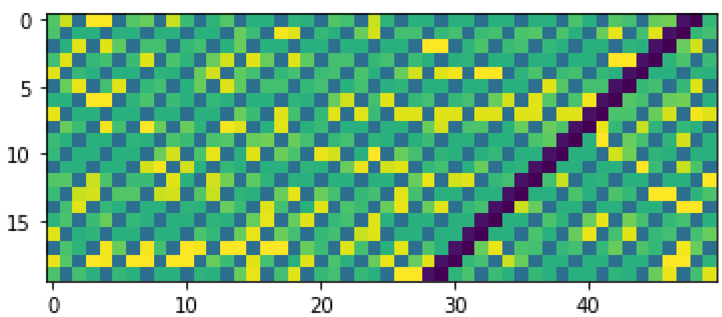



```python
# converting .asm file to byte file and use first 1000 bytes of each asm file as image filefiles=os.listdir("asmFiles/")

Y=pd.read_csv("trainLabels.csv")
filenames=Y['Id'].tolist()

filenames2=[]
image_feat_bytes=[]

number_of_feature = 1000
bytes_file_name= os.listdir("byteFiles/")

for file in filenames:
    filenames2.append(file)
    if(file+".txt" in bytes_file_name):
        with open("byteFiles/"+file+".txt", mode='rb') as f: # b is important -> binary
            fileContent = f.read()
```

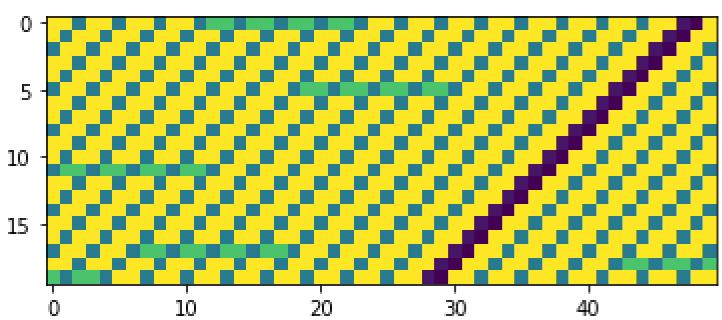
```
            top_1000_img_feat=(fileContent[:number_of_feature])

            image_feat_bytes.append(np.frombuffer(top_1000_img_feat, dtype = np
.uint8))
```

```
for i in range(5):
    print(f"example of byte image : {i+1}")
    plt.imshow(image_feat_bytes[i].reshape(20,50))
    plt.show()
```

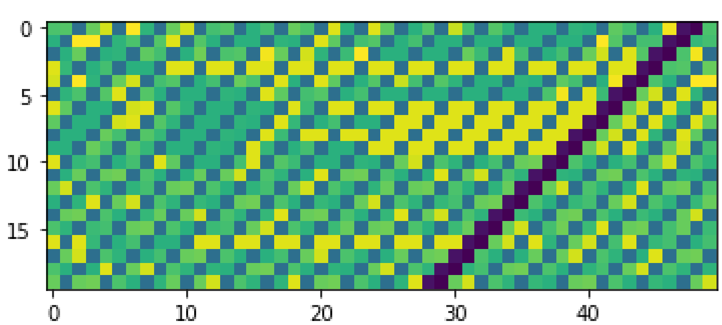
```
example of byte image : 1
```



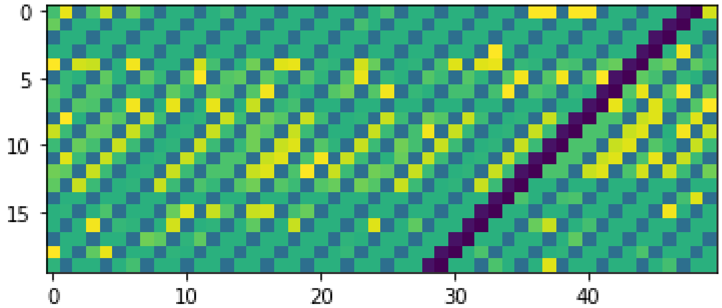
```
example of byte image : 2
```



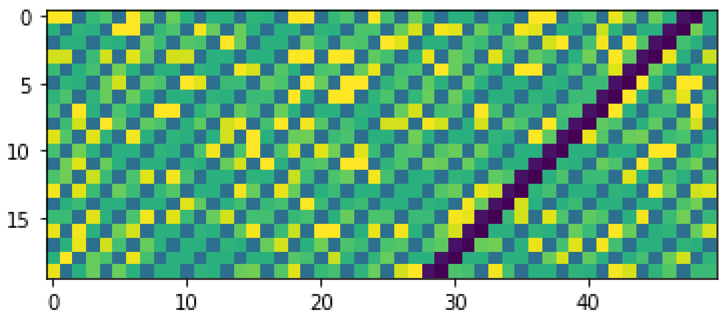
```
example of byte image : 3
```



```
example of byte image : 4
```



```
example of byte image : 5
```



```
from sklearn.preprocessing import normalize
image_intesity_feat=[np.hstack((image_feat_asm[i],image_feat_bytes[i])) for i i
n range(10868)]
image_intesity_feat = normalize(image_intesity_feat)
```

```
# Creating Dataframe of image feature
filenames=Y['Id'].tolist()
image_dataframe=pd.DataFrame()

image_dataframe["ID"]=filenames
for i in range(2000):
    im=[]
    for j in range(len(image_intesity_feat)):
        im.append(image_intesity_feat[j][i])
    image_dataframe["img_feat"+str(i)]=im

image_dataframe.head()
```

	ID	img_feat0	img_feat1	img_feat2	img_feat3	img_feat4	img_feat
0	01kcPWA9K2BOxQeS5Rju	0.028747	0.027549	0.025952	0.027150	0.027549	0.032739
1	04EjIdbPV5e1XroFOpiN	0.027361	0.026221	0.024701	0.025841	0.026221	0.031161
2	05EeG39MTRrI6VY21DPd	0.029153	0.027938	0.026319	0.027533	0.027938	0.033202
3	05rJTUWYAKNegBk2wE8X	0.028815	0.027614	0.026013	0.027214	0.027614	0.032817
4	0AnoOZDNbPXIr2MRBSCJ	0.028623	0.027431	0.025840	0.027033	0.027431	0.032599

5 rows × 2001 columns

```
# saving IMAGE FEATURE  into pickle
"""image_dataframe.to_pickle("image_dataframe.pkl")"""
image_dataframe = pd.read_pickle("image_dataframe.pkl")
```

5.2.1 Modling using Only Image Features

Train Test split

```
data_y = Y.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(image_dataframe.drop(["ID"
],axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same dist
ribution 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_tra
in,test_size=0.20)
```

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

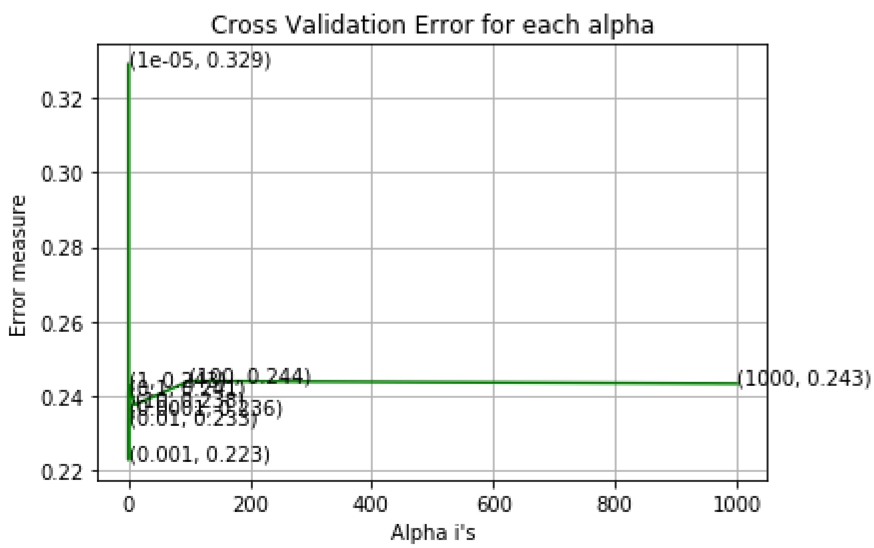
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

5.2.2. Logistic Regression

```
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',n_job
s=-2)
    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.classe
s_, 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='bal
```

```
anced',n_jobs=-2)
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.classe
s_, 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.cl
asses_, eps=1e-15))
```

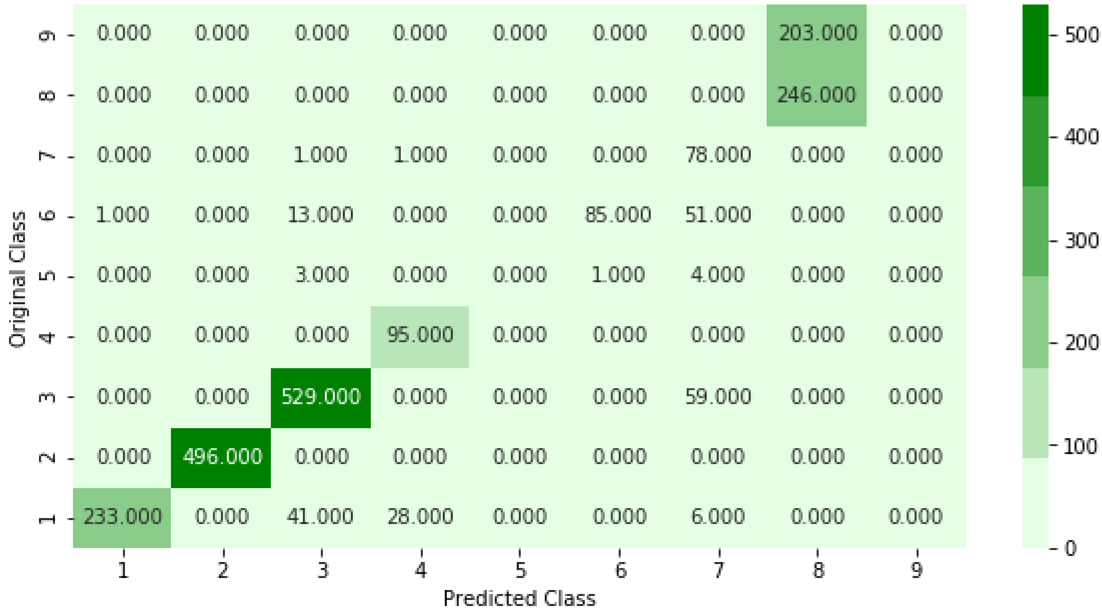
```
log_loss for c =  1e-05 is 0.32894689856609066
log_loss for c =  0.0001 is 0.2356014635922713
log_loss for c =  0.001 is 0.2229489609451024
log_loss for c =  0.01 is 0.2328894513559603
log_loss for c =  0.1 is 0.24095153463390018
log_loss for c =  1 is 0.24293803732920174
log_loss for c =  10 is 0.23765015624275335
log_loss for c =  100 is 0.24405178235738278
log_loss for c =  1000 is 0.24335958944979305
```



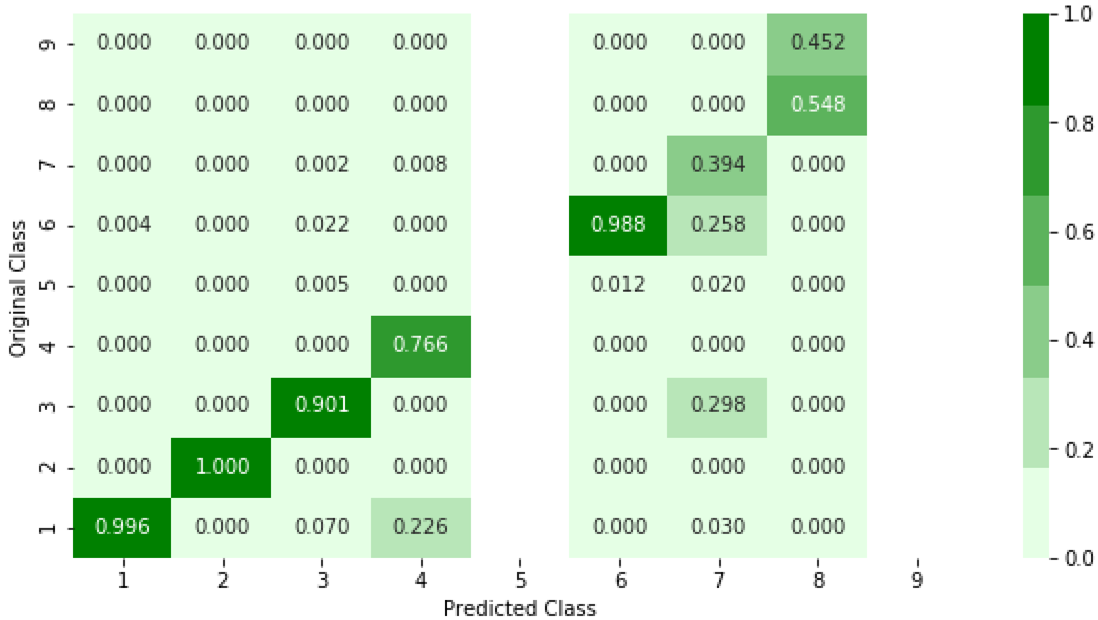

```
log loss for train data 0.134685563462311
log loss for cv data 0.2229489609451024
log loss for test data 0.2514524028445926
```

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

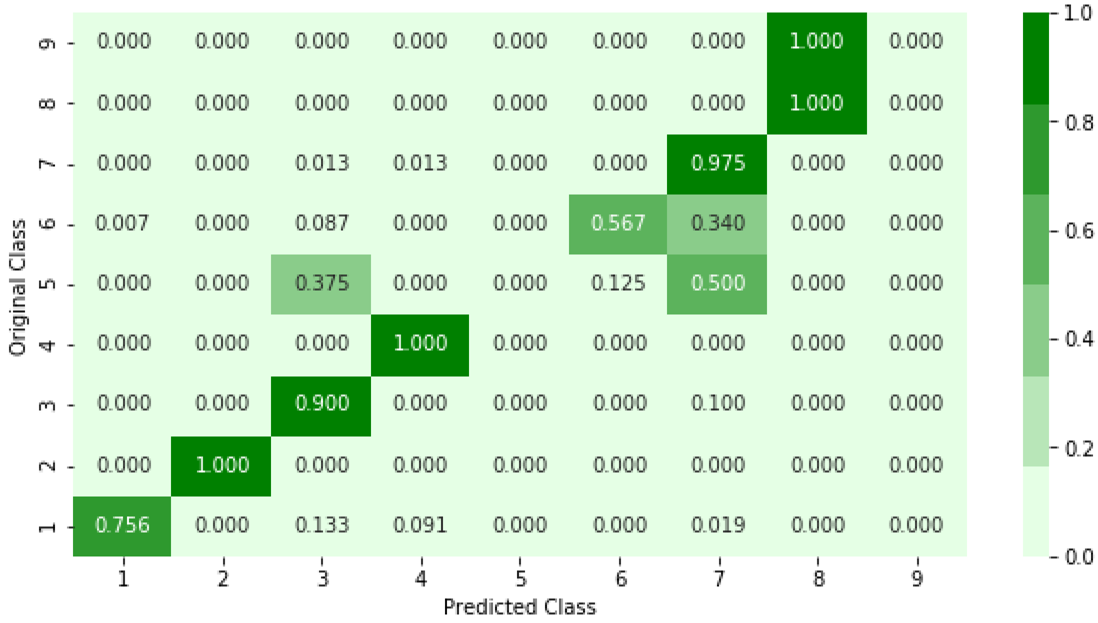
```
Number of misclassified points  18.951241950321986
-------------------------------------------------- Confusion matrix ------------------------
-------------------------
```




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




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



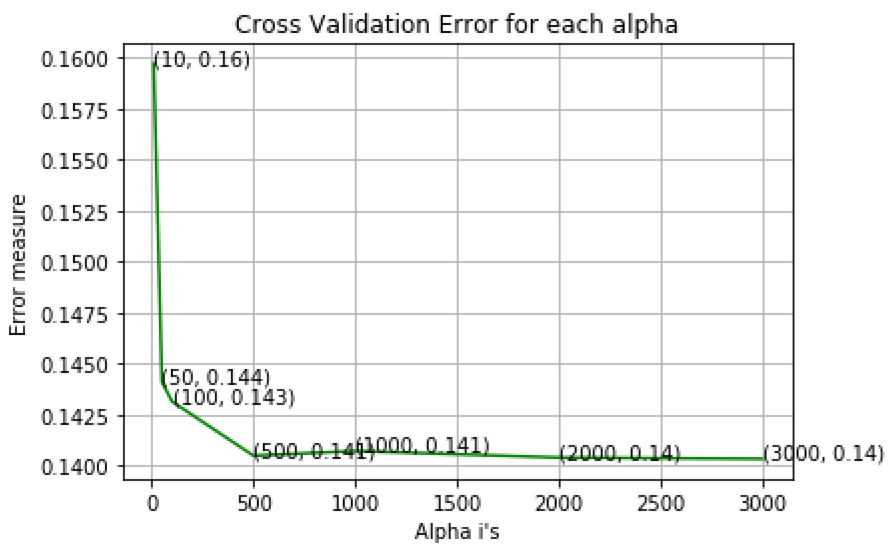

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

5.2.3 Random Forest Classifier

```
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_j
obs=-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 i
s:",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 l
og loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:"
,log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.15972058184371327
log_loss for c =  50 is 0.14411791050460487
log_loss for c =  100 is 0.1431597800031789
log_loss for c =  500 is 0.14050469897249862
log_loss for c =  1000 is 0.14073419175555982
log_loss for c =  2000 is 0.1404132093808151
log_loss for c =  3000 is 0.1403498307071034
```




```
For values of best alpha =  3000 The train log loss is: 0.047758595100963155
For values of best alpha =  3000 The cross validation log loss is: 0.1403498307071034
For values of best alpha =  3000 The test log loss is: 0.15046640100188594
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-55-820a381b4578> in <module>
     39 predict_y = sig_clf.predict_proba(X_test)
     40 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
_loss(y_test, predict_y))
---> 41 plot_confusion_matrix(y_test, sig_clf.predict(X_test))

NameError: name 'plot_confusion_matrix' is not defined
```

5.3 Traning on (ByteUnigram + asm + image feature

```
)
```

```
# merging feature
result_bytes_asm_size = pd.merge( left=result_x ,right=image_dataframe ,how='left',on='ID')
result_all_features= pd.merge( left=result_bytes_asm_size ,right=Y ,how='left',left_on='ID',right_on="ID")

result_all_features.head()
```

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148

5 rows × 2309 columns

Train Test Split of all features

```
#class_label
data_y = result_all_features.Class
#Input data
data_x=result_all_features.drop(["ID","Class"], axis=1)

# 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(data_x, data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

```
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])
```

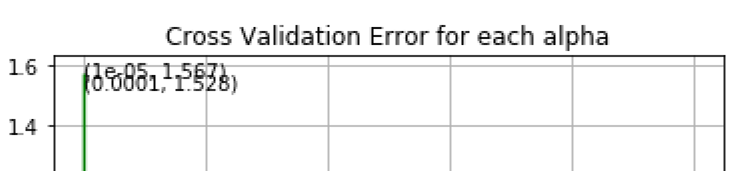
```
Number of data points in train data: 6724
Number of data points in test data: 2102
Number of data points in cross validation data: 1682
```

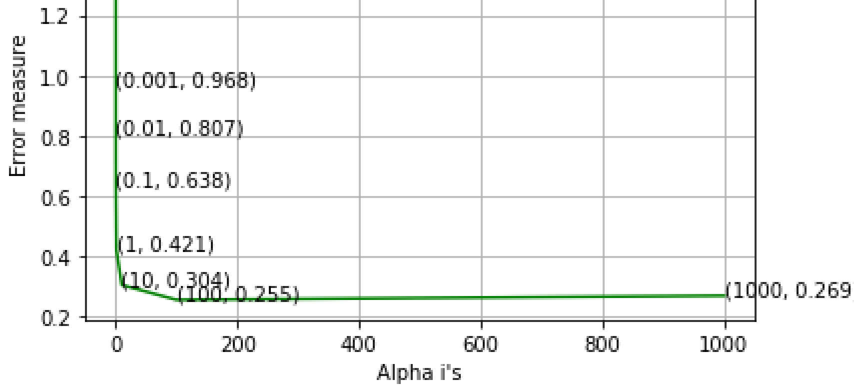
5.3.1. Logistic Regression

```
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',n_jobs=-2)
    logisticR.fit(X_train,y_train)
```

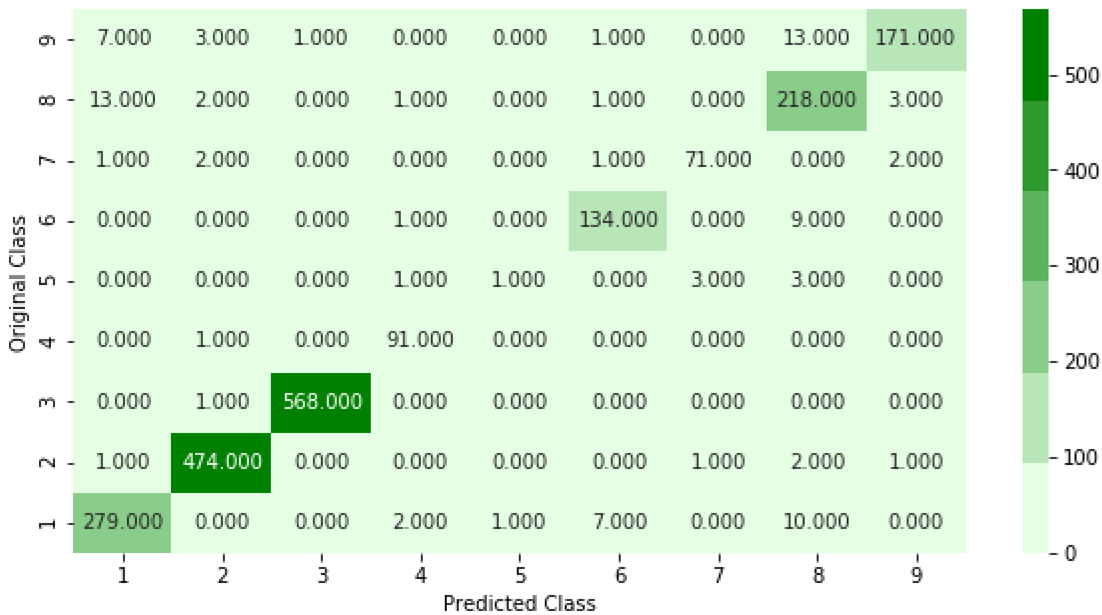
```
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classe
s_, eps=1e-15))

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='bal
anced',n_jobs=-2)
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.classe
s_, 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.cl
asses_, eps=1e-15))

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

```
log_loss for c =  1e-05 is 1.5673117034399247
log_loss for c =  0.0001 is 1.527712542208374
log_loss for c =  0.001 is 0.968067273009148
log_loss for c =  0.01 is 0.8066697709521494
log_loss for c =  0.1 is 0.6375480933709708
log_loss for c =  1 is 0.4214555740626545
log_loss for c =  10 is 0.30435327832875275
log_loss for c =  100 is 0.255143907228403
log_loss for c =  1000 is 0.2688696977042007
```

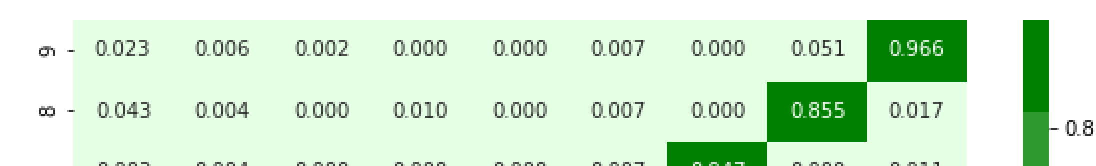


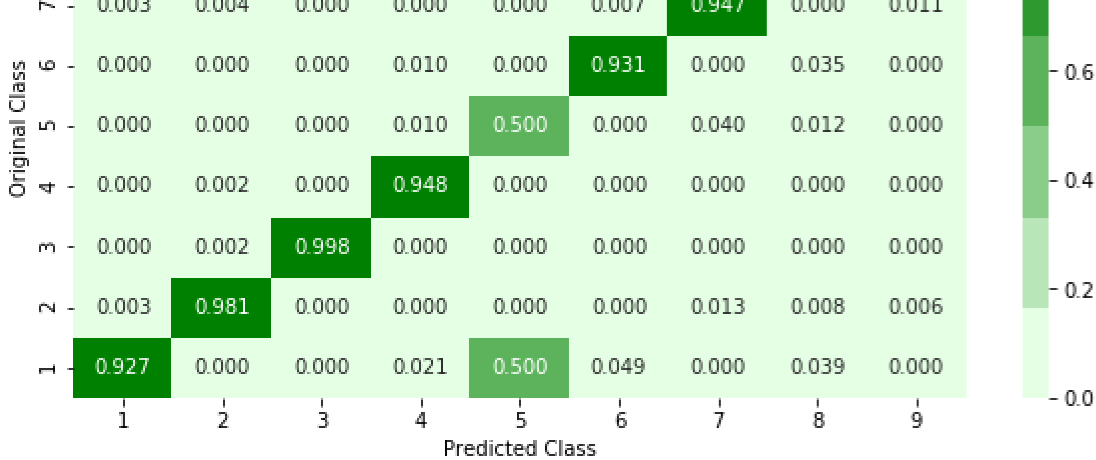



```
log loss for train data 0.20023153572904398
log loss for cv data 0.255143907228403
log loss for test data 0.20960673746305325
Number of misclassified points  4.519505233111323
-------------------------------------------------- Confusion matrix ------------------------
--------------------------
```

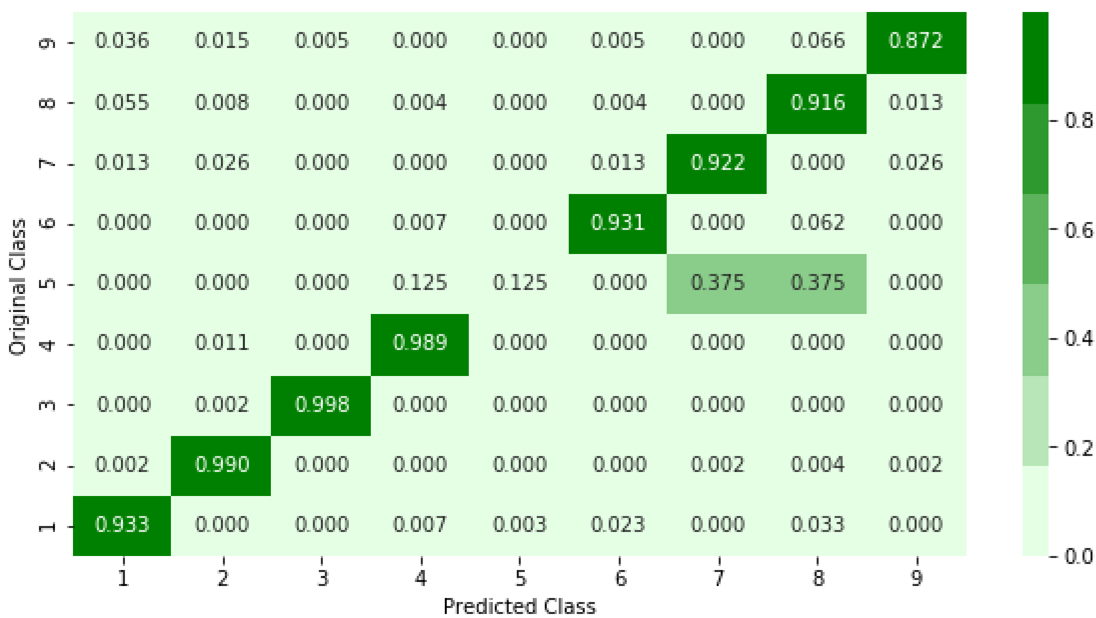



```
-------------------------------------------------- Precision matrix ------------------------
--------------------------
```






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



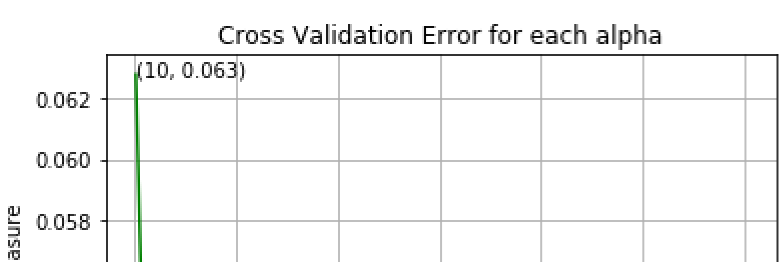

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

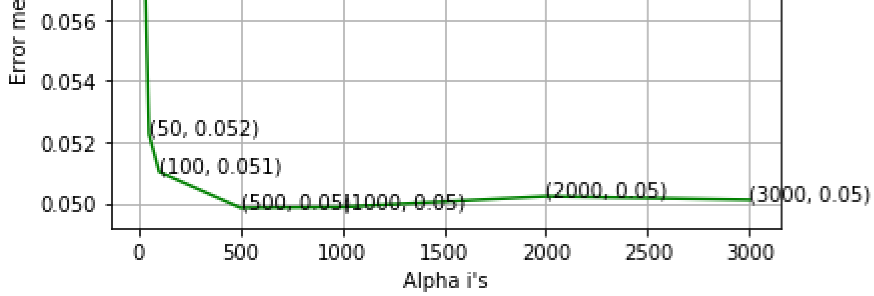
5.3.2 Random Forest Classifier

```
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")
```

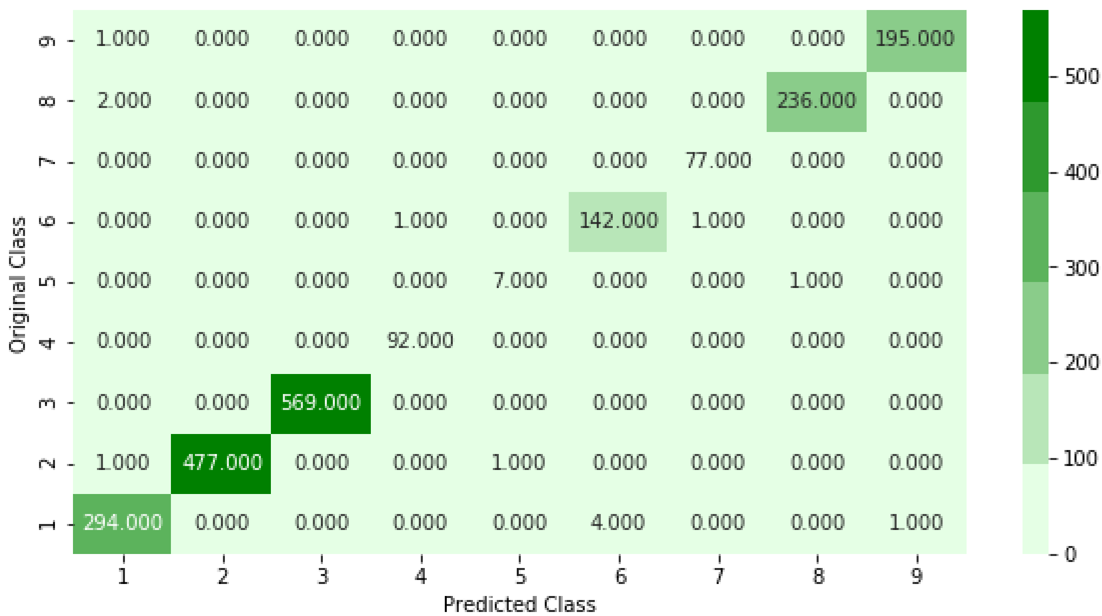
```
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_,
eps=1e-15))

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_j
obs=-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 i
s:",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 l
og loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:"
,log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.06276586880865438
log_loss for c =  50 is 0.05225726626566666
log_loss for c =  100 is 0.05102759799095187
log_loss for c =  500 is 0.0498521395477448
log_loss for c =  1000 is 0.04988022505205593 6
log_loss for c =  2000 is 0.05023069041972032
log_loss for c =  3000 is 0.05011643954466757
```

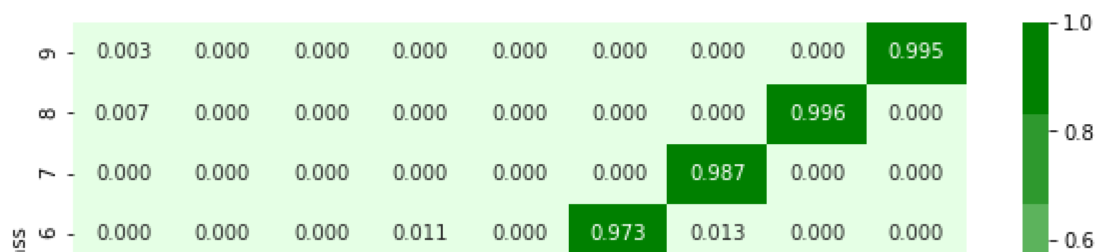




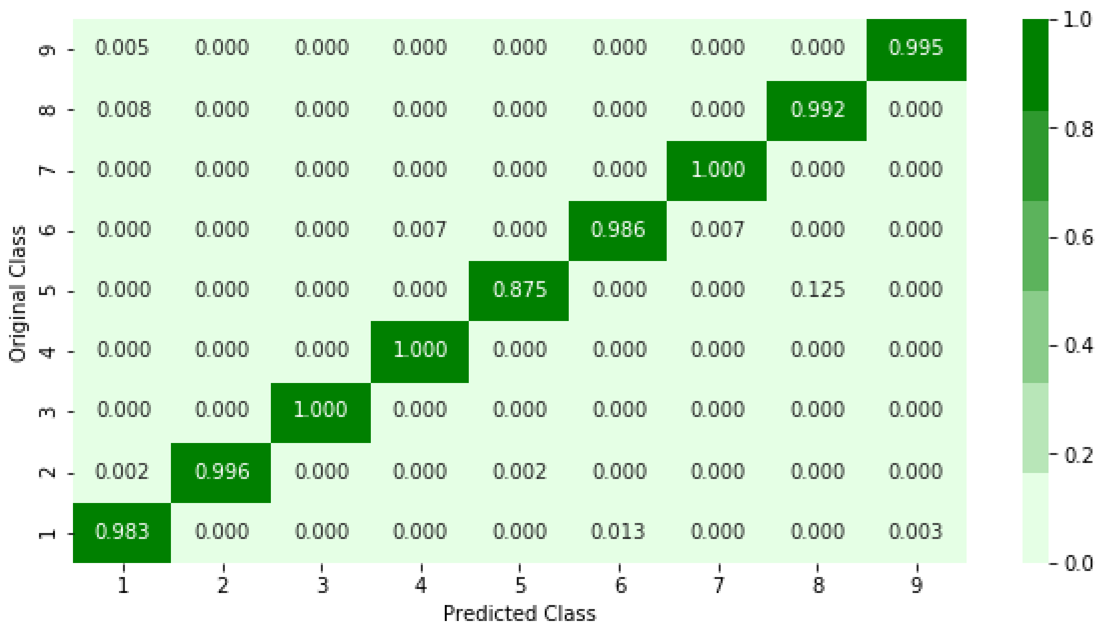

```
For values of best alpha =  500 The train log loss is: 0.01725951682301977
For values of best alpha =  500 The cross validation log loss is: 0.0498521395477448
For values of best alpha =  500 The test log loss is: 0.034633014003552766
Number of misclassified points  0.6184586108468125
---------------------------------------------- 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.]
```

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

```
x_cfl=XGBClassifier()

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

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

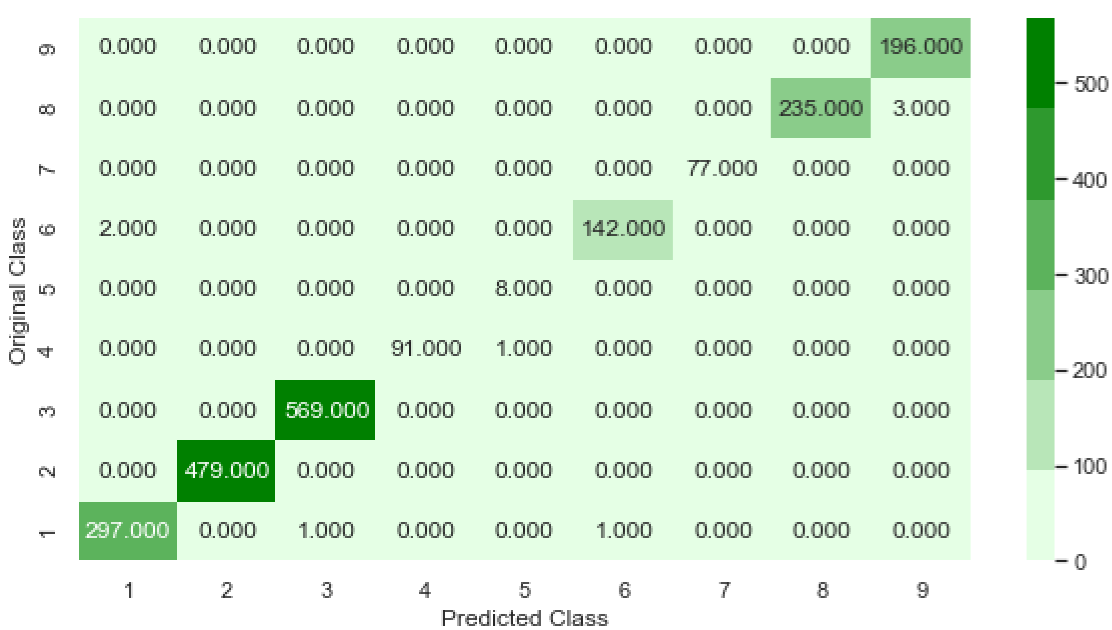
```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.2, 'colsample_by
tree': 0.5}
```

```
random_cfl.best_score_
```

```
-0.023188168339218634
```

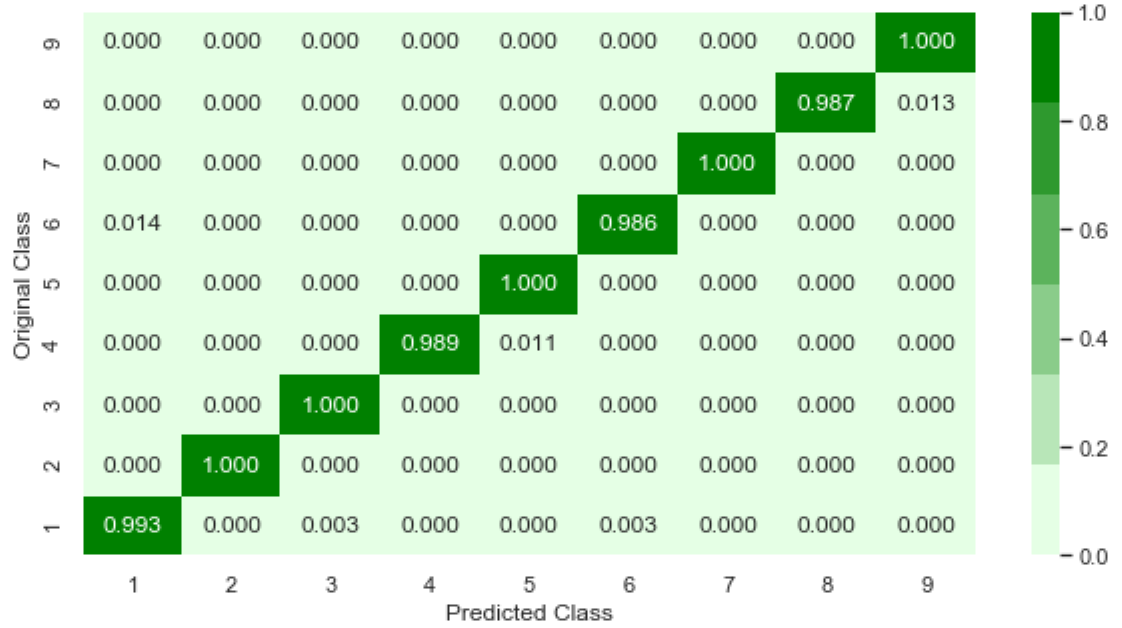
```
# Traning Using Best Hyper Parameter
x_cfl=XGBClassifier(n_estimators=2000,max_depth=3,learning_rate=0.2,colsample_b
ytree=0.5,subsample=0.5,nthread=-1)
x_cfl.fit(X_train, y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)



predict_y = sig_clf.predict_proba(X_train)

print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, sig_clf.predict_proba
(X_cv)))
predict_y = sig_clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test,sig_clf.predict_proba(X_test)))

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

```
The train log loss is: 0.010436344978044848
The cross validation log loss is: 0.021752886986359046
The test log loss is: 0.0245494878830332
Number of misclassified points  0.3805899143672693
---------------------------------------------------- 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.]
```

5.4. Stacking model (ASM + ByteUnigram + ByteBigram + Imagefeature)

1. Performing Logistic regression on Byte unigram because of high dimention
2. And performing XGBoost on remaining features
3. After that concatenate all the predited output and perform Final Model on the top of that (I am using XGBost as final model)

5.4.1. Modeling on Bytes Bigram feaures

```
#storing in pickle file after normalising bigram byte file
"""result_bigram.to_pickle("result_bigram.pkl")"""
result_bigram=pd.read_pickle("result_bigram.pkl")
```

Train Test split

```
# cleaning '.txt'  from the end of 'Id' names
ID_bigram = result_bigram.ID.apply(lambda x : x.strip(".txt"))

result_bigram.ID = ID_bigram
result_bigram.head()
```

	ID	????	0	1	2	3	4	5
0	01azqd4InC7m9JpocGv5	0.000129	0.230535	0.012146	0.010006	0.013938	0.008828	0.015486
1	01IsoiSMh5gxyDYTl4CB	0.000606	0.009218	0.041190	0.001032	0.001853	0.003406	0.000538
2	01jsnpXSAlgw6aPeDxrU	0.000033	0.009290	0.035909	0.000318	0.000403	0.026276	0.043230
3	01kcPWA9K2BOxQeS5Rju	0.000984	0.004490	0.006601	0.002621	0.002578	0.002711	0.002581
4	01SuzwMJEIXsK7A8dQbl	0.000636	0.007179	0.001320	0.001191	0.004189	0.000278	0.000645

```
5 rows × 65538 columns
```

```
# merging with class label on 'ID'
Y=pd.read_csv("trainLabels.csv")
result_bigram= result_bigram.merge(right=Y ,how='inner',left_on='ID',right_on=
'Id')

# class label
data_y = result_bigram['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_bigram.drop(['ID','C
lass','Id'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same dist
ribution 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_tra
in,test_size=0.20)
```

```
alpha[best_alpha]=500
r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_j
obs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

bigram_X_train = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(y_train, bigram_X_train))
bigram_X_cv = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation l
og loss is:",log_loss(y_cv, bigram_X_cv))
bigram_X_test = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:"
,log_loss(y_test, bigram_X_test))
```

```
For values of best alpha =  500 The train log loss is: 0.023561847893255806
For values of best alpha =  500 The cross validation log loss is: 0.06442365850161981
For values of best alpha =  500 The test log loss is: 0.0720668108530215
```

5.4.2. XGBoost on Remaining feature (Byte unigram + asm +size +image feat of ASM files only)

```
# merging feature
result_bytes_asm_size = pd.merge( left=result_x ,right=image_dataframe ,how='le
ft',on='ID')
result_all_features= pd.merge( left=result_bytes_asm_size ,right=Y ,how='left',
left_on='ID',right_on="Id")

result_all_features.head()
```

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058

1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	(
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	(
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	(
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	(

5 rows × 1309 columns

Train Test Split of all features

```
#class_label
data_y = result_all_features.Class
#Input data
data_x=result_all_features.drop(["ID","Id","Class"], axis=1)

# 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(data_x, data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

```
x_cfl=XGBClassifier(n_estimators=500,max_depth=3,learning_rate=0.03,colsample_bytree=0.3,subsample=0.5,nthread=-1)
x_cfl.fit(X_train, y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)


remaining_X_train = sig_clf.predict_proba(X_train)
print ("The train log loss is:",log_loss(y_train, remaining_X_train))
remaining_X_cv = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, remaining_X_cv))
remaining_X_test = sig_clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, remaining_X_test))
```

```
The train log loss is: 0.010835106688873318
The cross validation log loss is: 0.03135665679167791
The test log loss is: 0.03258715796342816
```

```
bigram_X_train
```

```
array([[1.08338507e-03, 9.89740130e-01, 5.42190264e-04, ...,
        1.87850717e-03, 3.47953746e-03, 1.05368664e-03],
       [2.87384113e-03, 2.53761659e-03, 6.21708253e-04, ...,
        1.94253893e-03, 4.98373118e-03, 5.13782617e-03],
       [9.73784917e-04, 2.23355727e-03, 9.88052165e-01, ...,
        1.84820277e-03, 3.38606685e-03, 1.11956343e-03],
       ...,
```

```
 [1.32626728e-03, 2.44509303e-03, 6.12648739e-04, ...,
  1.86308527e-03, 5.83697829e-03, 1.35443547e-03],
 [9.75407820e-04, 9.89931224e-01, 5.42317130e-04, ...,
  1.84956024e-03, 3.44137066e-03, 1.04137256e-03],
 [7.57634169e-02, 1.05195450e-02, 6.68599104e-04, ...,
  2.52771020e-03, 8.94490641e-01, 8.07053886e-03]])
```

5.4.3. Concating of output of models and traning using XGBoost

```
# Concating of output of models
X_train_final = np.hstack((bigram_X_train, remaining_X_train))
X_cv_final = np.hstack((bigram_X_cv, remaining_X_cv))
X_test_final = np.hstack((bigram_X_test, remaining_X_test))
```

```
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,scoring='neg_log_loss')
random_cfl.fit(X_train_final, y_train)
```

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   17.9s
[Parallel(n_jobs=-1)]: Done  11 out of  30 | elapsed:   40.1s remaining:  1.2min
[Parallel(n_jobs=-1)]: Done  15 out of  30 | elapsed:   41.4s remaining:   41.4s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:   43.7s remaining:   25.3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:   45.0s remaining:   13.6s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:   56.5s remaining:    6.2s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.0min finished
```

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_al...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.15, 0.2],
```

```
                                     'max_depth': [3, 5, 10],
                                     'n_estimators': [100, 200, 500, 1000,
                                                      2000],
                                     'subsample': [0.1, 0.3, 0.5, 1]},
                  pre_dispatch='2*n_jobs', random_state=None, refit=True,
                  return_train_score=False, scoring='neg_log_loss',
                  verbose=10)
```

```
random_cfl.best_params_
```

```
{'subsample': 1,
 'n_estimators': 2000,
 'max_depth': 10,
 'learning_rate': 0.01,
 'colsample_bytree': 0.5}
```

```
x_cfl=XGBClassifier(n_estimators=2000,max_depth=10,learning_rate=0.01,colsample
_bytree=0.5,subsample=1,nthread=-1)
x_cfl.fit(X_train_final, y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_final, y_train)


train_pred = sig_clf.predict_proba(X_train_final)
print ("The train log loss is:",log_loss(y_train, train_pred))
cv_pred = sig_clf.predict_proba(X_cv_final)
print( "The cross validation log loss is:",log_loss(y_cv, cv_pred))
test_pred = sig_clf.predict_proba(X_test_final)
print("The test log loss is:",log_loss(y_test, test_pred))
```

```
The train log loss is: 0.004469629438830589
The cross validation log loss is: 0.040390044816735124
The test log loss is: 0.036003412139572456
```

5.5 Merging all the feature

1. Choosing top features of BytesBigram using RandomForest.
2. And then merging top bytes bigram features with other features.
3. Train it using various models

5.5.1. Choosing top features of bigram

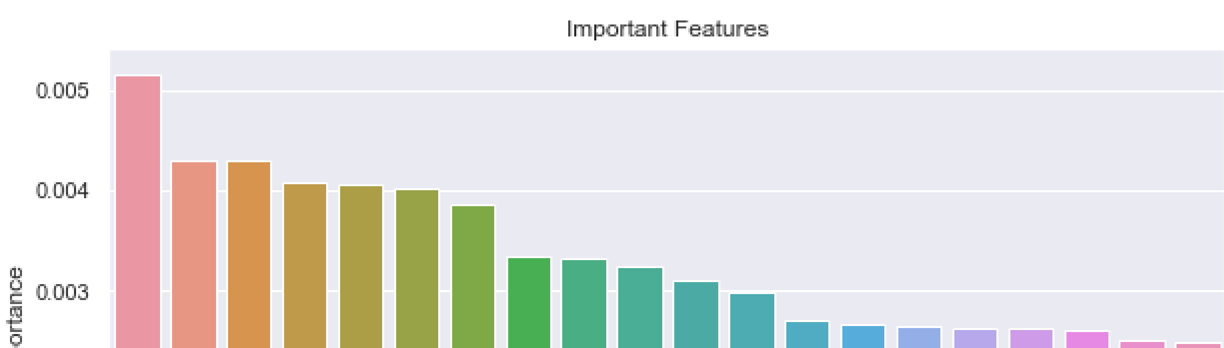
```
## Choosing top features of bigram

id_name = result_bigram.ID.apply(lambda x: x.strip(".txt").strip())
result_bigram.ID= id_name
x=result_bigram.merge(right=Y ,how='inner',left_on='ID',right_on='ID')

result_y= x.Class
x=x.drop(["Class","ID"],axis=1)
x.head()
```

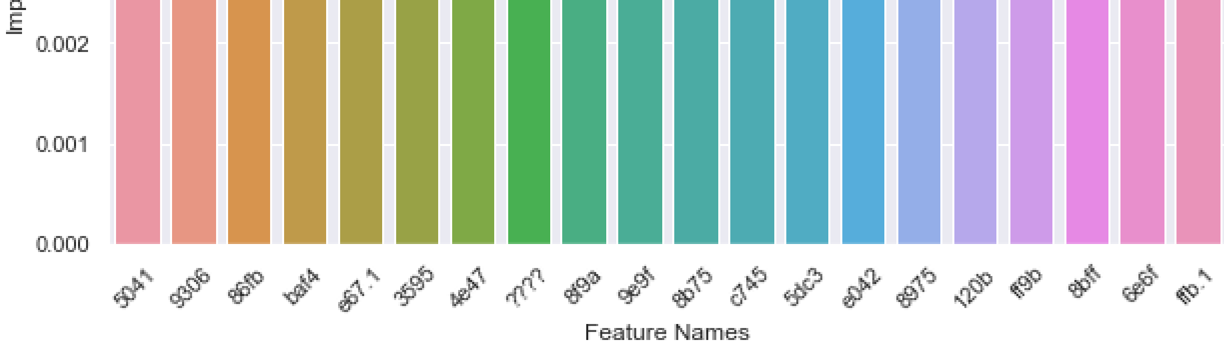
	????	0	1	2	3	4	5	6	7
0	0.000129	0.230535	0.012146	0.010006	0.013938	0.008828	0.015486	0.013424	0.015913	0.0085

1	0.000606	0.009218	0.041190	0.001032	0.001853	0.003406	0.000538	0.000102	0.000379	0.0010
2	0.000033	0.009290	0.035909	0.000318	0.000403	0.026276	0.043230	0.040781	0.000379	0.0251
3	0.000984	0.004490	0.006601	0.002621	0.002578	0.002711	0.002581	0.000915	0.001263	0.0021
4	0.000636	0.007179	0.001320	0.001191	0.004189	0.000278	0.000645	0.000102	0.000505	0.0002

5 rows × 65537 columns

```
# Function to choose top bigram features using Random Forest Model
# reference:  https://github.com/sai977/microsoft-malware-detection/blob/maste
r/MicrosoftMalwareDetection.ipynb
def imp_features(data, features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    #result_y=
    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]

byte_bi_indxes = imp_features(x, x.columns, 1000)
```






```
top_bigram_feat=x.iloc[:,byte_bi_indxes]
top_bigram_feat["ID"]=id_name
top_bigram_feat.head()
```

	5041	9306	86fb	baf4	e67.1	3595	4e47	????	8f9a	9e9f
0	0.005771	0.000068	0.000938	0.02500	0.025	0.021429	0.009174	0.000129	0.010870	0.018416
1	0.000000	0.000000	0.000000	0.00000	0.000	0.000000	0.006881	0.000606	0.000000	0.007366
2	0.004946	0.000058	0.001126	0.02500	0.025	0.064286	0.006881	0.000033	0.032609	0.009208
3	0.019786	0.000000	0.000375	0.00625	0.000	0.000000	0.016055	0.000984	0.000000	0.003683
4	0.018137	0.000000	0.000000	0.00625	0.000	0.000000	0.000000	0.000636	0.005435	0.001842

5 rows × 1001 columns

```
merged_feat= pd.merge(top_bigram_feat,result_x,on="ID")
merged_feat= pd.merge(merged_feat,image_dataframe,on='ID')
merged_feat= pd.merge(merged_feat, Y,left_on="ID",right_on='ID')
```

```
merged_feat.head()
```

	5041	9306	86fb	baf4	e67.1	3595	4e47	????	8f9a	9e9f
0	0.005771	0.000068	0.000938	0.02500	0.025	0.021429	0.009174	0.000129	0.010870	0.018416
1	0.000000	0.000000	0.000000	0.00000	0.000	0.000000	0.006881	0.000606	0.000000	0.007366
2	0.004946	0.000058	0.001126	0.02500	0.025	0.064286	0.006881	0.000033	0.032609	0.009208
3	0.019786	0.000000	0.000375	0.00625	0.000	0.000000	0.016055	0.000984	0.000000	0.003683
4	0.018137	0.000000	0.000000	0.00625	0.000	0.000000	0.000000	0.000636	0.005435	0.001842

5 rows × 3309 columns

```
# saving all the features into pickle file
"""
merged_feat.to_pickle("merged_feat.pkl")
"""

# Loading 'result_all_features' using pickle
merged_feat=pd.read_pickle("merged_feat.pkl")
```

Train Test Split of all features

```
#class_label
```

```
data_y = merged_feat.Class
#Input data
data_x=merged_feat.drop(["ID","Class"], axis=1)

# 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(data_x, data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

```
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])
```

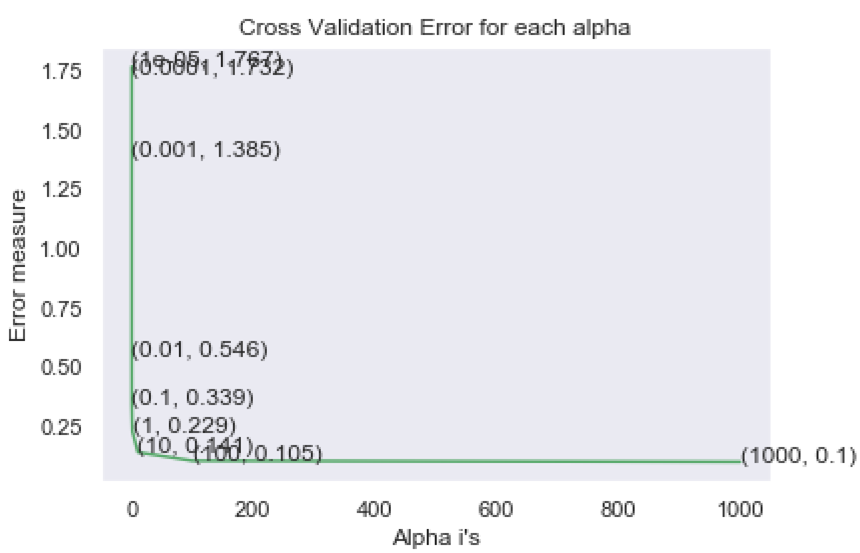
```
Number of data points in train data: 6502
Number of data points in test data: 2033
Number of data points in cross validation data: 1626
```

5.5.2. Logistic Regression

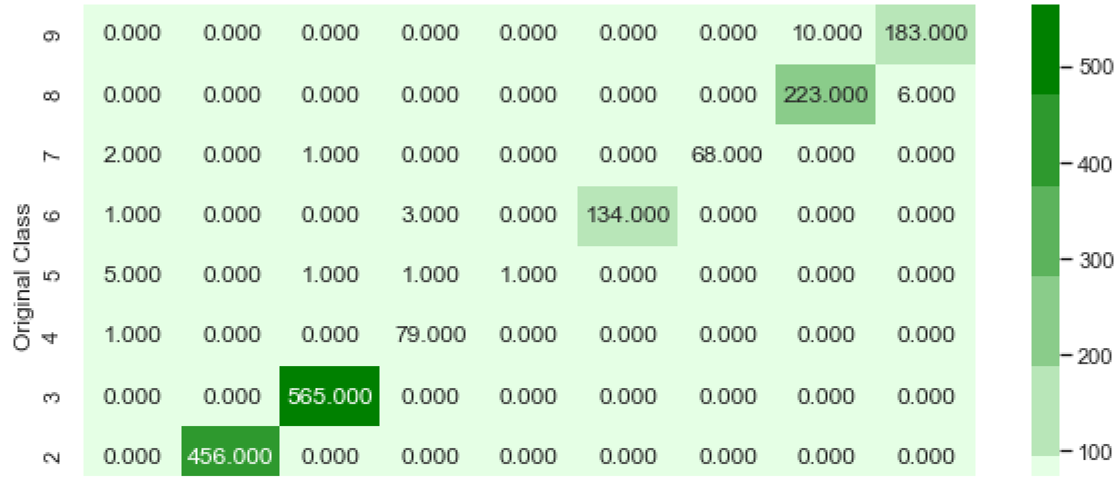
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced',n_jobs=-2)
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    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',n_jobs=-2)
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.classe
s_, 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.cl
asses_, eps=1e-15))

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

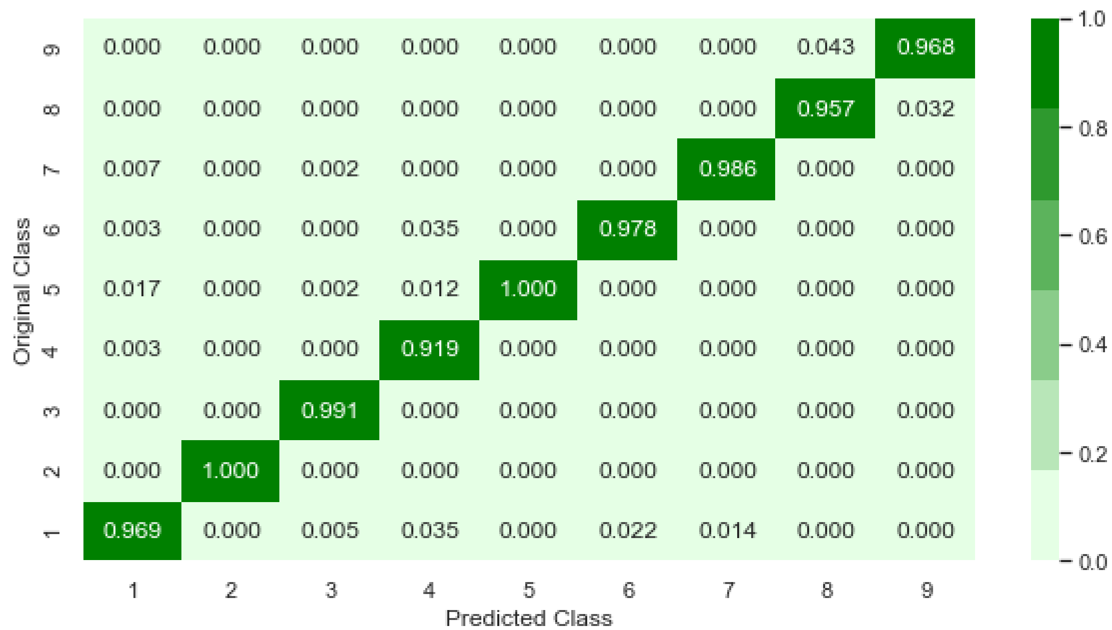
```
log_loss for c =  1e-05 is 1.7671806652741804
log_loss for c =  0.0001 is 1.7315409601838954
log_loss for c =  0.001 is 1.3847578080053684
log_loss for c =  0.01 is 0.5460749883516517
log_loss for c =  0.1 is 0.3394411809439536
log_loss for c =  1 is 0.22891853829380465
log_loss for c =  10 is 0.1406399683903932
log_loss for c =  100 is 0.1045884208639312
log_loss for c =  1000 is 0.10012564824980222
```



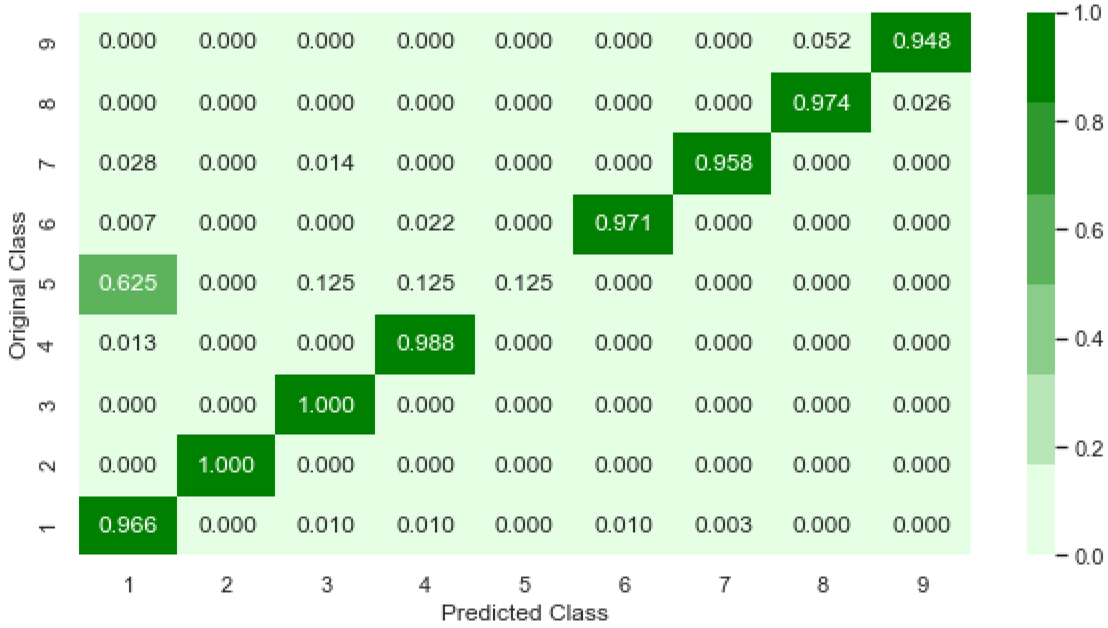

```
log loss for train data 0.05234163165107671
log loss for cv data 0.10012564824980222
log loss for test data 0.10136234324222235
Number of misclassified points  2.016724053123463
-------------------------------------------------- Confusion matrix ------------------------
--------------------------
```




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




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



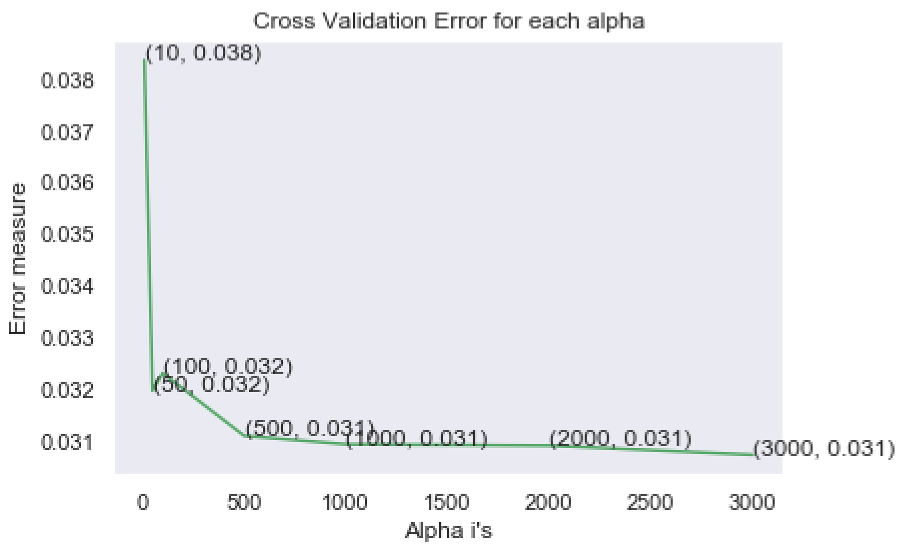

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

5.5.3 Random Forest Classifier

```
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_j
```

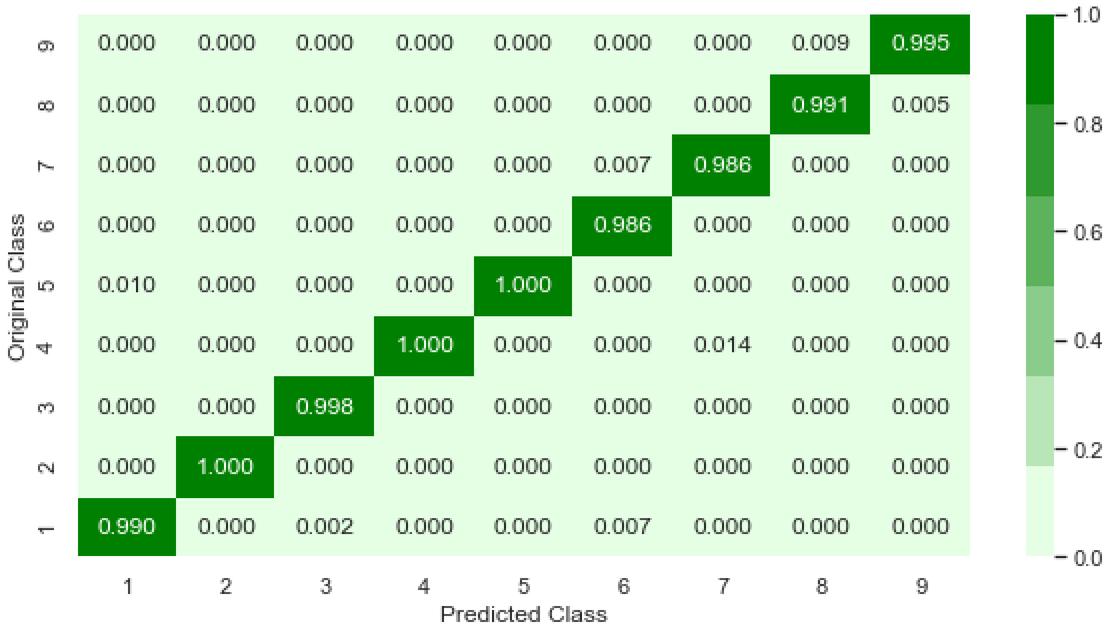
```
obs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
log_loss for c =  10 is 0.03837140204707954
log_loss for c =  50 is 0.03195826622141354
log_loss for c =  100 is 0.03230148548361279
log_loss for c =  500 is 0.0310884059677916
log_loss for c =  1000 is 0.030924334871007562
log_loss for c =  2000 is 0.03089880073114503
log_loss for c =  3000 is 0.030720650219137374
```




```
For values of best alpha =  3000 The train log loss is: 0.014159915033890011
For values of best alpha =  3000 The cross validation log loss is: 0.030720650219137374
For values of best alpha =  3000 The test log loss is: 0.024050326483615176
Number of misclassified points  0.49188391539596654
-------------------------------------------------- Confusion matrix ------------------------
--------------------------
```

-- Precision matrix --



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

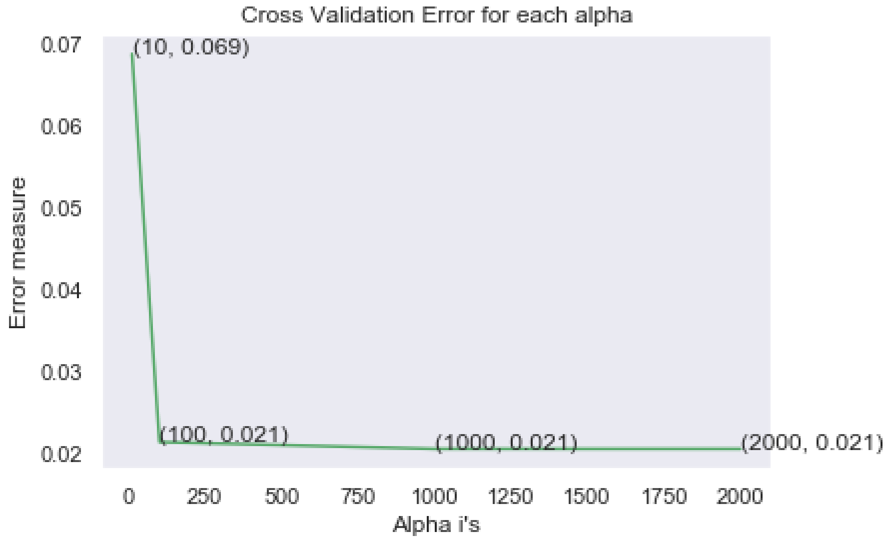

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

5.5.4. XGBoost Classifier

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

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


best_alpha = np.argmin(cv_log_error_array)

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

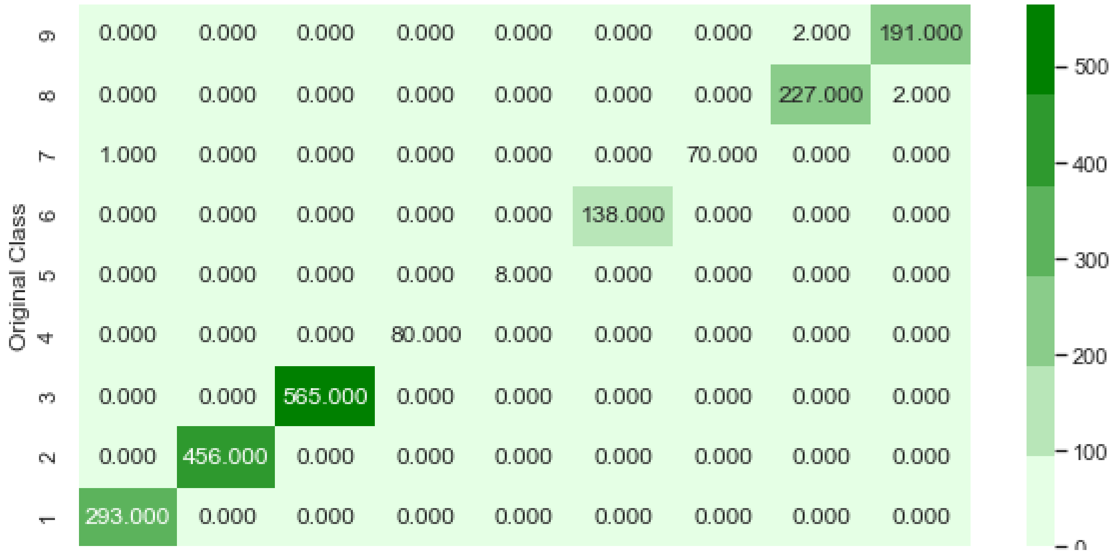
```
plt.ylabel("Error measure")
plt.show()
```

```
log_loss for c =  10 is 0.0687597763554462
log_loss for c =  100 is 0.021442481732995955
log_loss for c =  1000 is 0.0206089354381312
log_loss for c =  2000 is 0.0206081619215304
```

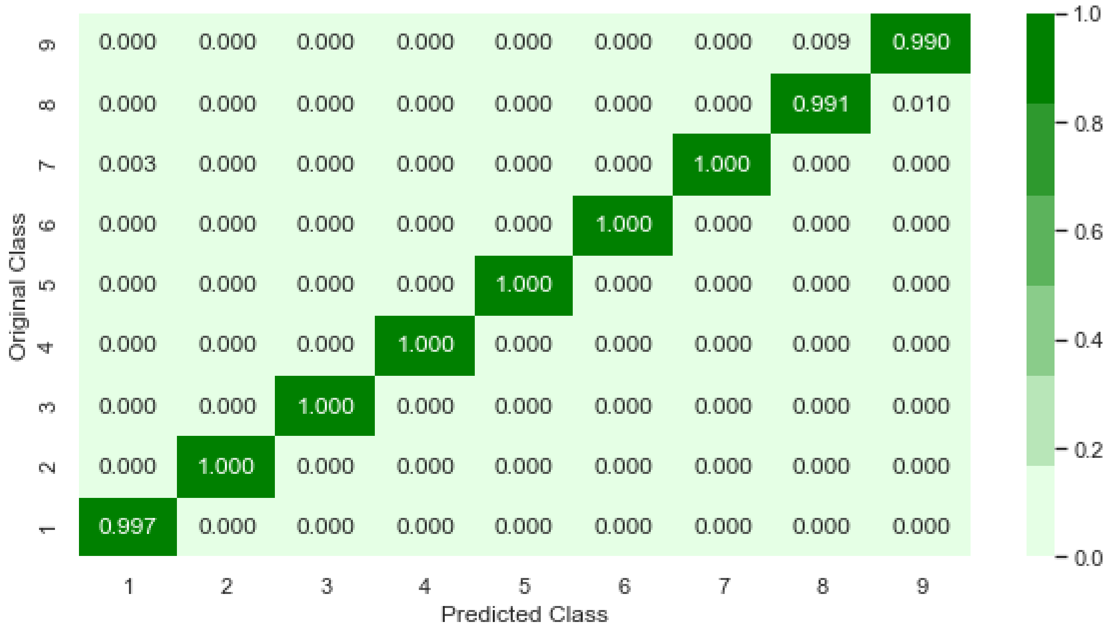



```
# Traning Usimg Optimal hyperparameter
# -----------------------------------

x_cfl=XGBClassifier(n_estimators=1000)
x_cfl.fit(X_train, y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)

print ("The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print( "The cross validation log loss is:",log_loss(y_cv, sig_clf.predict_proba
(X_cv)))
predict_y = sig_clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test,sig_clf.predict_proba(X_test)))

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

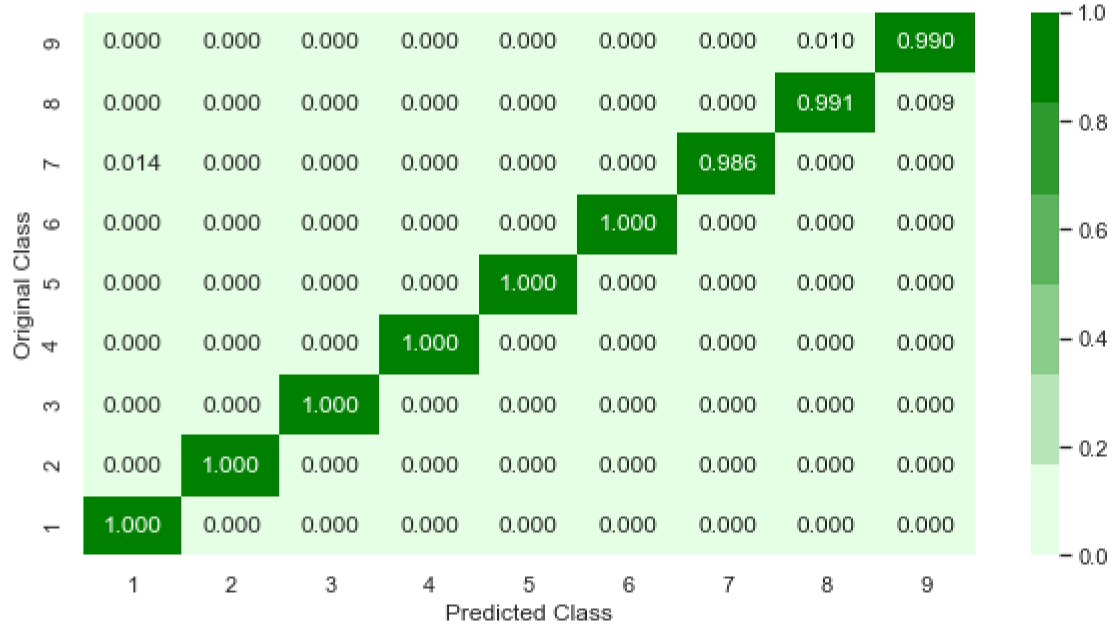
```
The train log loss is: 0.010206553438818185
The cross validation log loss is: 0.02786530575641078
The test log loss is: 0.01521340274414629
Number of misclassified points  0.24594195769798327
-------------------------------------------------- Confusion matrix ------------------------
-------------------------
```




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

6. Result and Conclusion

```
from prettytable import PrettyTable
print("RESULTS")

#===========================================================================
======================
print("="*100)
print("Feature: None  Model: Random")
print("="*100, )
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["NONE", "Random_model", 2.48, 2.48, 2.48 ])
print(x)

#===========================================================================
======================
print("\n")
print("="*100)
print("Feature : Byte_unigram ")
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["Byte_unigram", "KNN", 0.126, 0.211, 0.237 ])
x.add_row(["Byte_unigram", "Logistic Reg", 0.50, 0.531, 0.553 ])
x.add_row(["Byte_unigram", "Random Forest", 0.030, 0.084, 0.075 ])
x.add_row(["Byte_unigram", "Xgboost", 0.026, 0.064, 0.064 ])
x.add_row(["Byte_unigram", "Xgboost hypertuned", 0.026, 0.068, 0.066 ])
print(x)

#===========================================================================
======================
print("\n")
print("="*100)
print("Feature : ASM_unigram ")
```

```
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["ASM_unigram", "KNN", 0.024, 0.101, 0.09 ])
x.add_row(["ASM_unigram", "Logistic Reg",0.32, 0.367,0.33 ])
x.add_row(["ASM_unigram", "Random Forest", 0.011, 0.036, 0.030 ])
x.add_row(["ASM_unigram", "Xgboost", 0.010, 0.027, 0.030 ])
x.add_row(["ASM_unigram", "Xgboost hypertuned", 0.009, 0.031, 0.026 ])
print(x)
```

```
RESULTS
====================================================================================================
Feature: None  Model: Random
====================================================================================================
+----------+--------------+------------+---------+-----------+
| Features |    Model     | train_loss | CV_loss | Test_loss |
+----------+--------------+------------+---------+-----------+
|   NONE   | Random_model |    2.48    |   2.48  |    2.48   |
+----------+--------------+------------+---------+-----------+


====================================================================================================
Feature : Byte_unigram
====================================================================================================
+--------------+--------------------+------------+---------+-----------+
|   Features   |       Model        | train_loss | CV_loss | Test_loss |
+--------------+--------------------+------------+---------+-----------+
| Byte_unigram |        KNN         |   0.126    |  0.211  |   0.237   |
| Byte_unigram |    Logistic Reg    |    0.5     |  0.531  |   0.553   |
| Byte_unigram |   Random Forest    |    0.03    |  0.084  |   0.075   |
| Byte_unigram |      Xgboost       |   0.026    |  0.064  |   0.064   |
| Byte_unigram | Xgboost hypertuned |   0.026    |  0.068  |   0.066   |
+--------------+--------------------+------------+---------+-----------+


====================================================================================================
Feature : ASM_unigram
====================================================================================================
+-------------+--------------------+------------+---------+-----------+
|   Features  |       Model        | train_loss | CV_loss | Test_loss |
+-------------+--------------------+------------+---------+-----------+
| ASM_unigram |        KNN         |   0.024    |  0.101  |    0.09   |
| ASM_unigram |    Logistic Reg    |    0.32    |  0.367  |    0.33   |
| ASM_unigram |   Random Forest    |   0.011    |  0.036  |    0.03   |
| ASM_unigram |      Xgboost       |    0.01    |  0.027  |    0.03   |
| ASM_unigram | Xgboost hypertuned |   0.009    |  0.031  |   0.026   |
+-------------+--------------------+------------+---------+-----------+
```

```
print("ASSIGNMENT")

#=====================================================================================================
print("\n")
print("="*100)
```

```
print("Feature : byte_unigram + asm_unigram ")
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["byte_unigram + asm_unigram", "Random Forest", 0.017, 0.040, 0.034
])
x.add_row(["byte_unigram + asm_unigram", "Xgboost(hypertuned)", 0.012, 0.03, 0.
026 ])
print(x)
#============================================================================
======================

print("\n")
print("="*100)
print("Feature : byte_bigram ")
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["byte_bigram", "Logistic Reg", 0.06, 0.157, 0.183 ])
x.add_row(["byte_bigram", "Xgboost(hypertuned)", 0.02, 0.074, 0.068 ])
print(x)
#============================================================================
======================

print("\n")
print("="*100)
print("Feature : Image_feat ")
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["Image_feat", "Logistic Reg", 0.134, 0.222, 0.251 ])
x.add_row(["Image_feat", "Randm Forest", 0.04, 0.140, 0.150 ])
print(x)
#============================================================================
======================

print("\n")
print("="*100)
print("Feature : Byte_ungram + asm_unigram + Image_feat ")
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["Byte_ungram + asm_unigram + Image_feat", "Logistic Reg",0.200, 0.25
5, 0.209])
x.add_row(["Byte_ungram + asm_unigram + Image_feat", "Random Forest", 0.017, 0.
049, 0.034 ])
x.add_row(["Byte_ungram + asm_unigram + Image_feat", "Xgboost", 0.010, 0.021,
0.024 ])
print(x)
#============================================================================
======================

print("\n")
print("="*100)
print("Model: Ensembe_stacking   Feature: (asm + byte_unigram + byte_bigram + I
mgage_feat ) ")
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["asm + byte_unigram + byte_bigram + Imgage_feat", "Stacking Model",
```

```
0.004, 0.040, 0.036 ])
print(x)
#===========================================================================
=====================

print("\n")
print("="*100)
print("Feature: All Feature Merged")
print("(asm + byte_unigram + top_1000_byte_bigram + Imgage_feat)" )
print("="*100)
x = PrettyTable(["Features","Model", "train_loss", "CV_loss", "Test_loss"])
x.add_row(["All Feature Merged", "Logistic Reg", 0.052, 0.100, 0.101 ])
x.add_row(["All Feature Merged", "Randomn Forest", 0.014, 0.030, 0.024 ])
x.add_row(["All Feature Merged", "XgBoost", 0.010, 0.027, 0.015 ])
print(x)
#===========================================================================
=====================
```

```
ASSIGNMENT


============================================================================================
========
Feature : byte_unigram + asm_unigram
============================================================================================
========
+-----------------------------+---------------------+------------+---------+-----------+
|           Features          |        Model        | train_loss | CV_loss | Test_loss |
+-----------------------------+---------------------+------------+---------+-----------+
| byte_unigram + asm_unigram |    Random Forest    |   0.017    |   0.04  |   0.034   |
| byte_unigram + asm_unigram | Xgboost(hypertuned) |   0.012    |   0.03  |   0.026   |
+-----------------------------+---------------------+------------+---------+-----------+


============================================================================================
========
Feature : byte_bigram
============================================================================================
========
+-------------+---------------------+------------+---------+-----------+
|   Features  |        Model        | train_loss | CV_loss | Test_loss |
+-------------+---------------------+------------+---------+-----------+
| byte_bigram |     Logistic Reg    |    0.06    |  0.157  |   0.183   |
| byte_bigram | Xgboost(hypertuned) |    0.02    |  0.074  |   0.068   |

+-------------+---------------------+------------+---------+-----------+


============================================================================================
========
Feature : Image_feat
============================================================================================
========
+------------+--------------+------------+---------+-----------+
|  Features  |    Model     | train_loss | CV_loss | Test_loss |
+------------+--------------+------------+---------+-----------+
| Image_feat | Logistic Reg |   0.134    |  0.222  |   0.251   |
| Image_feat | Randm Forest |    0.04    |   0.14  |    0.15   |
+------------+--------------+------------+---------+-----------+
```

Feature : Byte_ungram + asm_unigram + Image_feat

Features	Model	train_loss	CV_loss	Test_loss
Byte_ungram + asm_unigram + Image_feat	Logistic Reg	0.2	0.255	0.209
Byte_ungram + asm_unigram + Image_feat	Random Forest	0.017	0.049	0.034
Byte_ungram + asm_unigram + Image_feat	Xgboost	0.01	0.021	0.024

Model: Ensembe_stacking   Feature: (asm + byte_unigram + byte_bigram + Imgage_feat)

Features	Model	train_loss	CV_loss	Test_loss
asm + byte_unigram + byte_bigram + Imgage_feat	Stacking Model	0.004	0.04	0.036

Feature: All Feature Merged
(asm + byte_unigram + top_1000_byte_bigram + Imgage_feat)

Features	Model	train_loss	CV_loss	Test_loss
All Feature Merged	Logistic Reg	0.052	0.1	0.101
All Feature Merged	Randomn Forest	0.014	0.03	0.024
All Feature Merged	Logistic Reg	0.01	0.027	0.015

Conclusion

1. Additional features like byteBigram and Image_feat of ByteFiles and ASMFiles are definately helpful in improving the model performance
2. Best result comes by merging all the features (asm + byte_unigram + top_1000_byte_bigram + Imgage_feat)

3. Best Result is :

   Feature: All Feature Merged(asm + byte_unigram + top_1000_byte_bigram + Imgage_feat)

> ```
>   |  Model: Xgboost  |    Train_log_loss: 0.01    |  CV_log_l
> oss: 0.027  |   Test_log_loss: 0.015   |
> ```

1. Misclassification percentage using best features is 0.245 %
2. Adding additional features like n_gram of Opcode feature would also be helpful in improving perormance but due to computationl strain I am leaving that part.

END :)