In [1]:

```
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 tqdm import tqdm
import scipy
```

In [2]:

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

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

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

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

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

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

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

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

Machine Learning Models

XGBoost

In [3]:

```
#Load the byte and asm unigrams dataframe
#byte_features_with_size = pd.read_csv('normalized_bf.csv')

result_x = pd.read_csv("result_x.csv")

#Load class labels
result_y = pd.read_csv("result_y.csv").drop('Unnamed: 0',axis=1)




#Load the ASM unigrams dataframe
asm_features_with_size = pd.read_csv('normalized_asm.csv')

#Load the byte bigram dataframe with reduced features
top1000_byte_bigram_df = pd.read_csv('byte_bigram_df_1000.csv')

#Load the ASM Opcodes bigram dataframe with reduced features
top1000_opcodes_bi_tri_df = pd.read_csv('op_trigram_1000.csv')

top1000_opcodes_bi_tetr_df = pd.read_csv('op_tetragram_1000.csv')

#Load the ASM Registers bigram dataframe with reduced features
top500_image_df = pd.read_csv('img_final_500.csv')
```

asm and byte unigrams + top1000_byte_bigrams+ top1000_opcodes_bi_trigrams+ top500 pixel_images

In [4]:

```
final_data = pd.concat([result_x,  top1000_byte_bigram_df, top1000_opcodes_bi_tri_df,
top500_image_df], axis = 1, join = 'inner')
```

In [7]:

```
final_data.head()
```

Out[7]:

Unnamed: 0	Unnamed: 0.1	0	1	2	3	4	5	6	7	...	pix491	pix492	pi

	Unnamed: 0	Unnamed: 0.1	0	1	2	3	4	5	6	7	...	pix491	pix492	pi
0	0	0.000000	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.002942	0.010763	0.01
1	1	0.000092	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.019285	0.003312	0.00
2	2	0.000184	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	...	0.002942	0.010763	0.01
3	3	0.000276	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	...	0.002942	0.010763	0.01
4	4	0.000368	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	...	0.002942	0.010763	0.01

5 rows × 2816 columns

In [8]:

```
final_data = final_data.drop(['Unnamed: 0', 'Unnamed: 0.1', 'ID'],axis = 1)
```

In [10]:

```
final_data.head()
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	...	pix490	pix491	pix49
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	0.002942	0.002942	0.01076
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	0.019285	0.019285	0.00331
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	0.002942	0.002942	0.01076
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	0.002942	0.002942	0.01076
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	0.002942	0.002942	0.01076

5 rows × 2807 columns

In [154]:

```
result_y
```

Out[154]:

	Class
0	9
1	2
2	9
3	1
4	8
...	...
10863	4
10864	4
10865	4
10866	4
10867	4

10868 rows × 1 columns

In [13]:

```
X = final_data
y = result_y
```

In [14]:

```
#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(X, y, stratify=y, test_size=0.20, random_state=
42)

#Split the train data into train and cross validation by maintaining same distribution of output v
araible '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,
random_state=42)
```

In [17]:

```
print(X_train.shape)
print(y_train.shape)
print(X_cv.shape)
print(y_cv.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(6955, 2807)
(6955, 1)
(1739, 2807)
(1739, 1)
(2174, 2807)
(2174, 1)
```

In [18]:

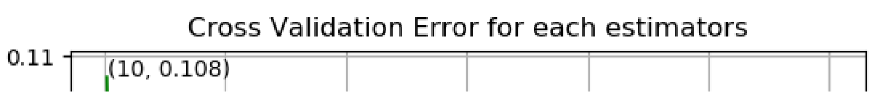
```
estimators=[10,50,100,250,500,1000,2000,3000]
cv_log_error_array=[]
for i in tqdm(estimators):
    x_cfl=XGBClassifier(n_estimators=i, n_jobs=-1)
    x_cfl.fit(X_train ,y_train )
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train , y_train )
    predict_y = sig_clf.predict_proba(X_cv )
    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 = ',estimators[i],'is',cv_log_error_array[i])
```

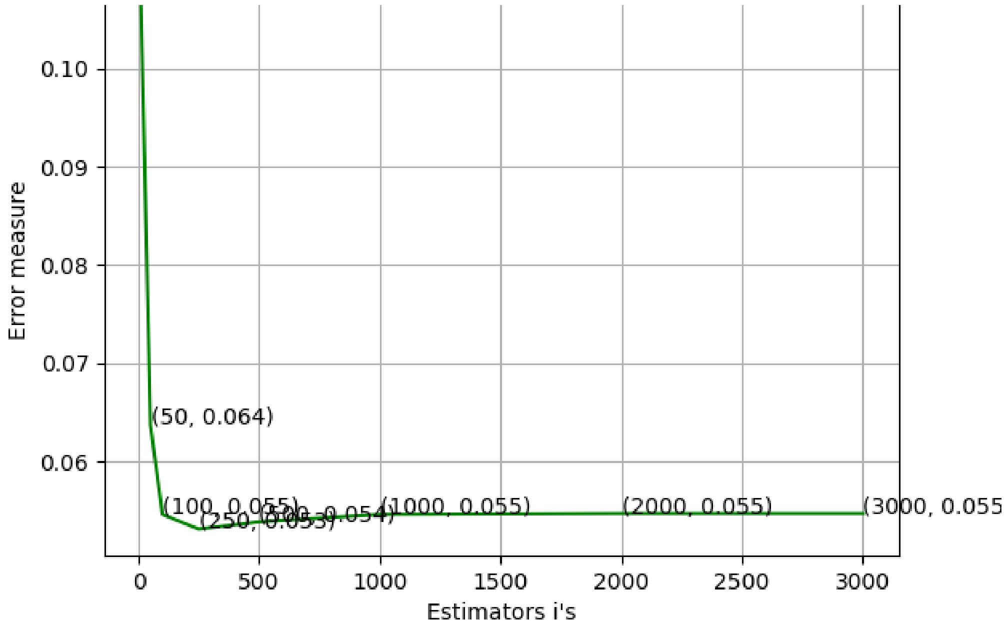
```
100%|██████████| 8/8 [1:33:47<00:00, 1193.17s/it]
```

```
log_loss for c =  10 is 0.10787841064572683
log_loss for c =  50 is 0.06371049825878322
log_loss for c =  100 is 0.05460506851459247
log_loss for c =  250 is 0.05312250887655023
log_loss for c =  500 is 0.053861746670928475
log_loss for c =  1000 is 0.05461130919526228
log_loss for c =  2000 is 0.05471279809571862
log_loss for c =  3000 is 0.05471184859370033
```

In [19]:

```
best_estimators = np.argmin(cv_log_error_array)

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





In [20]:

```
estimators[best_estimators]
```

Out[20]:

```
250
```

In [21]:

```
cv_log_error_array[best_estimators]
```

Out[21]:

```
0.05312250887655023
```

In [22]:

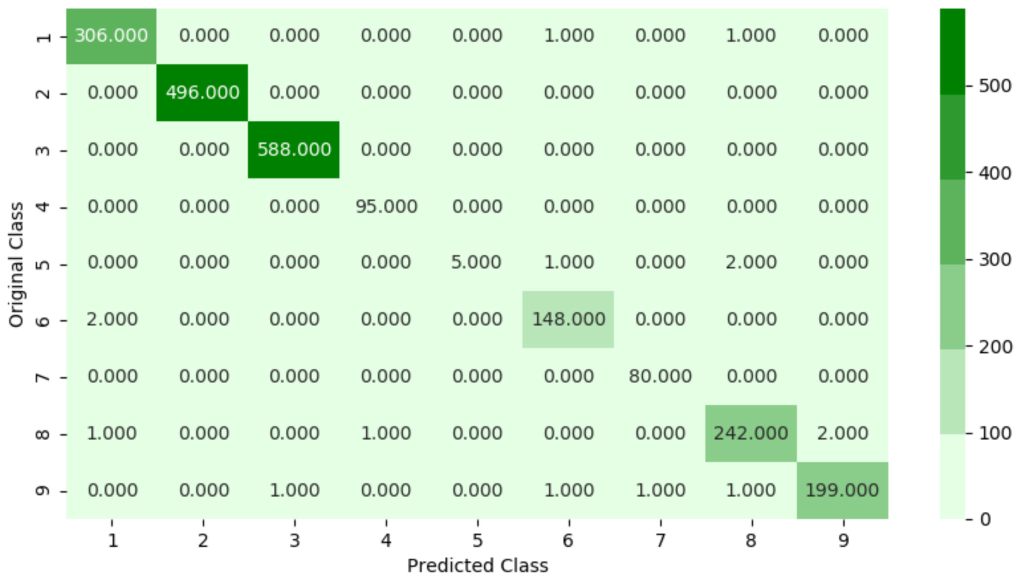
```
x_cfl=XGBClassifier(n_estimators=estimators[best_estimators],nthread=-1,n_jobs=-1)
x_cfl.fit(X_train ,y_train ,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train , y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best estimators = ', estimators[best_estimators], "The train log loss is:",l
og_loss(y_train , predict_y))
predict_y = sig_clf.predict_proba(X_cv )
print('For values of best estimators = ', estimators[best_estimators], "The cross validation log l
oss is:",log_loss(y_cv , predict_y))
predict_y = sig_clf.predict_proba(X_test )
print('For values of best estimators = ', estimators[best_estimators], "The test log loss is:",log
_loss(y_test , predict_y))
```

```
For values of best estimators =  250 The train log loss is: 0.012640251294804984
For values of best estimators =  250 The cross validation log loss is: 0.05312250887655023
For values of best estimators =  250 The test log loss is: 0.025342317026873618
```

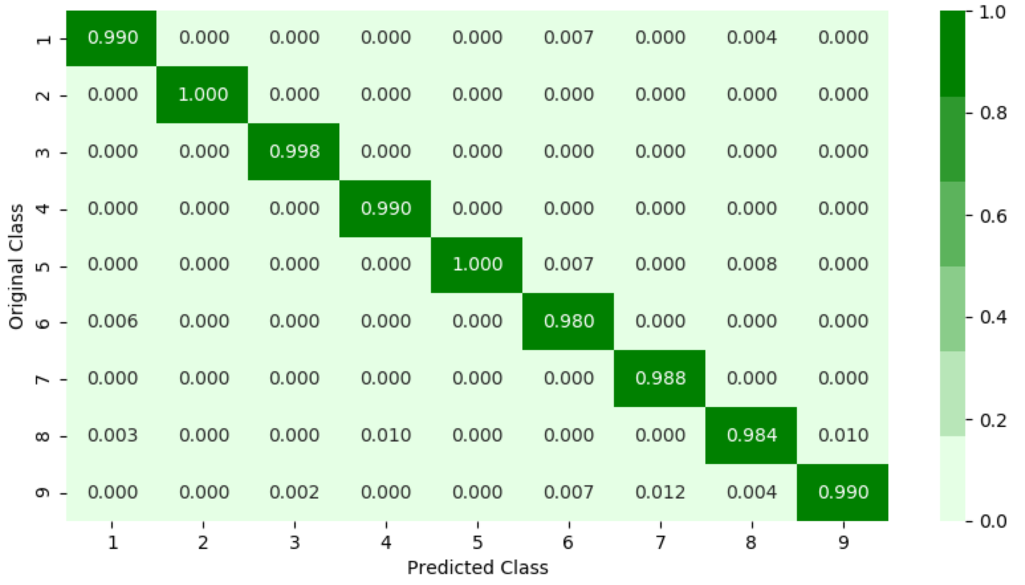
In [23]:

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

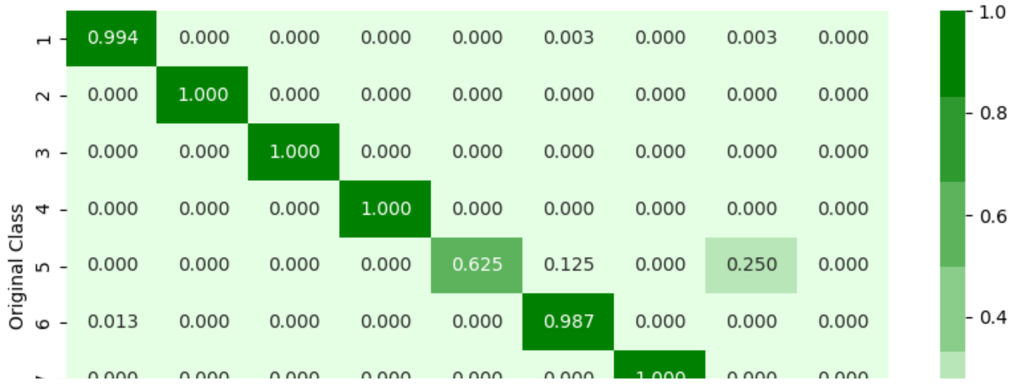
```
Number of misclassified points  0.6899724011039559
-------------------------------------------------- Confusion matrix -------------------------------
----------------
```

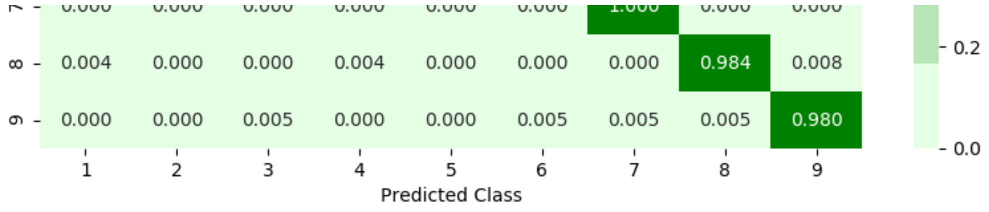


```
-------------------------------------------------- Precision matrix -------------------------------
----------------
```



```
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.]
```

So without any dimensionality reduction on the overall combined features on the final data,we get a test loss of 0.025

Let's see if we can reduce it further and get it down to 0.01 or less

In [24]:

```
from sklearn.decomposition import TruncatedSVD
```

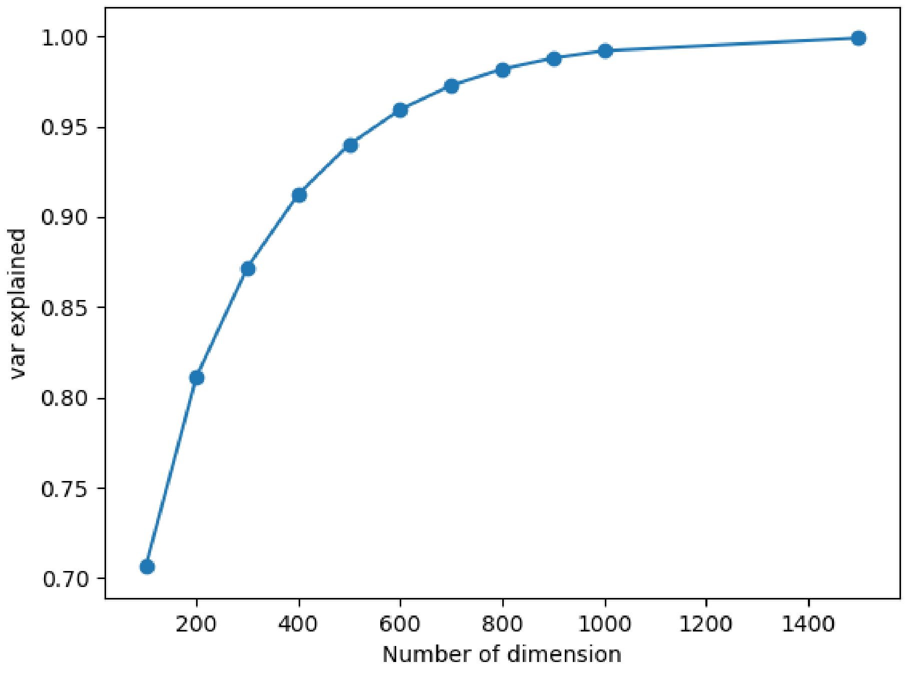
In [28]:

```
from datetime import datetime
var_explained=[]
start = datetime.now()
for i in [100,200,300,400,500,600,700,800,900,1000,1500]:
    print('Starting for i:',i)
    svd = TruncatedSVD(n_components=i, n_iter=5, random_state=42)
    svd.fit(X_train)
    var_explained.append(svd.explained_variance_ratio_.sum())
    print('Done for i:',i)
    print('Time taken for this particular i is:',datetime.now()-start)
```

```
Starting for i: 100
Done for i: 100
Time taken for this particular i is: 0:00:01.443318
Starting for i: 200
Done for i: 200
Time taken for this particular i is: 0:00:03.265701
Starting for i: 300
Done for i: 300
Time taken for this particular i is: 0:00:05.913099
Starting for i: 400
Done for i: 400
Time taken for this particular i is: 0:00:08.968343
Starting for i: 500
Done for i: 500
Time taken for this particular i is: 0:00:12.581358
Starting for i: 600
Done for i: 600
Time taken for this particular i is: 0:00:16.849652
Starting for i: 700
Done for i: 700
Time taken for this particular i is: 0:00:21.606551
Starting for i: 800
Done for i: 800
Time taken for this particular i is: 0:00:27.122552
Starting for i: 900
Done for i: 900
Time taken for this particular i is: 0:00:33.496492
Starting for i: 1000
Done for i: 1000
Time taken for this particular i is: 0:00:40.734446
Starting for i: 1500
Done for i: 1500
Time taken for this particular i is: 0:00:52.903794
```

In [32]:

```
plt.plot([100,200,300,400,500,600,700,800,900,1000,1500],var_explained)
plt.scatter([100,200,300,400,500,600,700,800,900,1000,1500],var_explained)
plt.xlabel('Number of dimension')
plt.ylabel('var explained')
plt.show()
```



In [35]:

```
start = datetime.now()
svd = TruncatedSVD(n_components=1500, n_iter=5, random_state=42)
svd.fit(X_train)
print('var explained :',svd.explained_variance_ratio_.sum())
#print('Done for i:',i)
print('Time taken for this:',datetime.now()-start)
```

```
var explained : 0.99898827666186
Time taken for this: 0:00:12.146113
```

- We observe that 1500 features explain most of the variance.
- So let's pick the top 1500 features using RandomForest and train the final model with reduced features

In [36]:

```
#we have now bring down the dimension to the 1500
X_train_1500 = svd.transform(X_train)
X_cv_1500 = svd.transform(X_cv)
X_test_1500 = svd.transform(X_test)
```

In [ ]:

```
### apply random forest on reduced features
```

In [37]:

```
alpha=[10,50,100,500,1000,2000,3000]
max_depth =[None,3,5,7,11,13,15]
cv_log_error_array=[]
for i in alpha:
    for j in tqdm(max_depth):
        r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1,max_depth=j)
        r_cfl.fit(X_train_1500,y_train)
        sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
        sig_clf.fit(X_train_1500, y_train)
        predict_y = sig_clf.predict_proba(X_cv_1500)
        cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))
        print('Done For alpha:',i,'depth',j)
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
 14%|█▍        | 1/7 [00:02<00:15,  2.55s/it]
```

```
Done For alpha: 10 depth None
```

```
 29%|██▊       | 2/7 [00:04<00:11,  2.25s/it]
```

```
Done For alpha: 10 depth 3
```

```
 43%|████▎     | 3/7 [00:06<00:08,  2.16s/it]
```

```
Done For alpha: 10 depth 5
```

```
 57%|█████▋    | 4/7 [00:08<00:06,  2.11s/it]
```

```
Done For alpha: 10 depth 7
```

```
 71%|███████▏  | 5/7 [00:10<00:04,  2.21s/it]
```

```
Done For alpha: 10 depth 11
```

```
 86%|████████▌ | 6/7 [00:12<00:02,  2.27s/it]
```

```
Done For alpha: 10 depth 13
```

```
100%|██████████| 7/7 [00:15<00:00,  2.30s/it]
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Done For alpha: 10 depth 15
```

```
 14%|█▍        | 1/7 [00:06<00:38,  6.34s/it]
```

```
Done For alpha: 50 depth None
```

```
 29%|██▊       | 2/7 [00:09<00:26,  5.34s/it]
```

```
Done For alpha: 50 depth 3
```

```
 43%|████▎     | 3/7 [00:13<00:19,  4.88s/it]
```

```
Done For alpha: 50 depth 5
```

```
 57%|█████▋    | 4/7 [00:17<00:14,  4.75s/it]
```

```
Done For alpha: 50 depth 7
```

```
 71%|███████▏  | 5/7 [00:23<00:09,  4.95s/it]
```

```
Done For alpha: 50 depth 11
```

```
 86%|████████▌ | 6/7 [00:29<00:05,  5.35s/it]
```

```
Done For alpha: 50 depth 13
```

```
100%|██████████| 7/7 [00:35<00:00,  5.67s/it]
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Done For alpha: 50 depth 15
```

```
 14%|█▍        | 1/7 [00:11<01:09, 11.64s/it]
```

```
Done For alpha: 100 depth None
```

```
 29%|██▊       | 2/7 [00:16<00:47,  9.55s/it]
```

```
Done For alpha: 100 depth 3
```

```
 43%|████▎     | 3/7 [00:22<00:34,  8.57s/it]
```

```
Done For alpha: 100 depth 5
```

```
 57%|█████▋    | 4/7 [00:30<00:24,  8.32s/it]
```

```
Done For alpha: 100 depth 7
```

```
 71%|███████▏  | 5/7 [00:40<00:17,  8.85s/it]
```

```
Done For alpha: 100 depth 11
```

```
 86%|████████▌ | 6/7 [00:51<00:09,  9.39s/it]
```

```
Done For alpha: 100 depth 13
```

```
100%|██████████| 7/7 [01:02<00:00,  9.92s/it]
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Done For alpha: 100 depth 15
```

```
 14%|█▍        | 1/7 [00:50<05:05, 50.85s/it]
```

```
Done For alpha: 500 depth None
```

```
 29%|██▊       | 2/7 [01:08<03:24, 40.98s/it]
```

```
Done For alpha: 500 depth 3
```

```
 43%|████▎     | 3/7 [01:34<02:25, 36.28s/it]
```

```
Done For alpha: 500 depth 5
```

```
 57%|█████▋    | 4/7 [02:06<01:45, 35.02s/it]
```

```
Done For alpha: 500 depth 7
```

```
 71%|███████▏  | 5/7 [02:47<01:13, 36.94s/it]
```

```
Done For alpha: 500 depth 11
```

```
 86%|████████▌ | 6/7 [03:33<00:39, 39.53s/it]
```

```
Done For alpha: 500 depth 13
```

```
100%|██████████| 7/7 [04:20<00:00, 41.93s/it]
  0%|          | 0/7 [00:00<?, ?it/s]
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Done For alpha: 500 depth 15
```

```
 14%|█▍        | 1/7 [01:35<09:34, 95.76s/it]
```

```
Done For alpha: 1000 depth None
```

```
 29%|██▊       | 2/7 [02:09<06:25, 77.17s/it]
```

```
Done For alpha: 1000 depth 3
```

```
 43%|████▎     | 3/7 [02:58<04:34, 68.67s/it]
```

```
Done For alpha: 1000 depth 5
```

```
 57%|█████▋    | 4/7 [04:01<03:20, 66.85s/it]
```

```
Done For alpha: 1000 depth 7
```

```
 71%|███████▏  | 5/7 [05:25<02:24, 72.05s/it]
```

```
Done For alpha: 1000 depth 11
```

```
 86%|████████▌ | 6/7 [06:56<01:17, 77.95s/it]
```

```
Done For alpha: 1000 depth 13
```

```
100%|██████████| 7/7 [08:32<00:00, 83.25s/it]
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Done For alpha: 1000 depth 15
```

```
 14%|█▍        | 1/7 [03:16<19:36, 196.08s/it]
```

```
Done For alpha: 2000 depth None
```

```
 29%|██▊       | 2/7 [04:19<13:01, 156.29s/it]
```

```
Done For alpha: 2000 depth 3
```

```
 43%|████▎     | 3/7 [05:57<09:15, 138.93s/it]
```

```
Done For alpha: 2000 depth 5
```

```
 57%|█████▋    | 4/7 [08:00<06:42, 134.02s/it]
```

```
Done For alpha: 2000 depth 7
```

```
 71%|███████▏  | 5/7 [10:47<04:47, 144.00s/it]
```

```
Done For alpha: 2000 depth 11
```

```
 86%|████████▌ | 6/7 [13:49<02:35, 155.41s/it]
```

```
Done For alpha: 2000 depth 13
```

```
100%|██████████| 7/7 [16:52<00:00, 163.57s/it]
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Done For alpha: 2000 depth 15
```

```
 14%|█         | 1/7 [04:42<28:17, 282.90s/it]
```

```
Done For alpha: 3000 depth None
```

```
 29%|██        | 2/7 [06:19<18:55, 227.05s/it]
```

```
Done For alpha: 3000 depth 3
```

```
 43%|████      | 3/7 [08:43<13:28, 202.21s/it]
```

```
Done For alpha: 3000 depth 5
```

```
 57%|█████     | 4/7 [11:49<09:51, 197.13s/it]
```

```
Done For alpha: 3000 depth 7
```

```
 71%|███████   | 5/7 [16:06<07:10, 215.28s/it]
```

```
Done For alpha: 3000 depth 11
```

```
 86%|████████  | 6/7 [20:34<03:50, 230.89s/it]
```

```
Done For alpha: 3000 depth 13
```

```
100%|██████████| 7/7 [25:13<00:00, 245.50s/it]
```

```
Done For alpha: 3000 depth 15
```

In [48]:

```
print(cv_log_error_array)
```

```
[0.5530674693257018, 0.8438928127711011, 0.6544299720768033, 0.5362313756704283, 0.5308908287285,
0.5216123350121442, 0.5330879331124206, 0.3895687682749189, 0.7393027760444636,
0.582685684052316, 0.4449193811556458, 0.3883529200770430З, 0.38542122157244446,
0.3886330957969547, 0.358543855881728, 0.7282555411063586, 0.5559969531911585, 0.4187653934971942,
0.36484272502170767, 0.3548689299949715, 0.35845072860343097, 0.32251372463699224,
0.6709671798804849, 0.5102789959215982, 0.40502363268064523, 0.3362363655599969,
0.3221119331812872, 0.3242780503228188, 0.31778831629789517, 0.6677435016021268,
0.5041018314174981, 0.40372874967147926, 0.33388755856177216, 0.3209891647798273,
0.32051901223234935, 0.3152559684428399, 0.6692606251366392, 0.502038252814857,
0.40198664913185933, 0.33211406123339215, 0.31997421189607356, 0.31629593804816836,
0.31375025049990746, 0.6681302090749879, 0.5003723961833979, 0.4012899329391691,
0.33128065352594616, 0.320210203587287, 0.3151971294432046]
```

In [49]:

```
min(cv_log_error_array)
```

Out[49]:

```
0.31375025049990746
```

ok this is not useful. Since the best cross validation error itself is quite high.i,e. 0.3

So let's not proceed with reduced features.

Let's take important features of the final data(all 2807 features) with randomforest

In [51]:

```
alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    print('Done for alpha:',i)
    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))
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

```
Done for alpha: 10
```

```
 17%|█         | 1/6 [00:02<00:10,  2.18s/it]
```

```
Done for alpha: 50
```

```
 33%|███       | 2/6 [00:06<00:10,  2.74s/it]
```

```
Done for alpha: 100
```

```
 50%|█████     | 3/6 [00:12<00:11,  3.75s/it]
```

```
Done for alpha: 500
```

```
 67%|██████    | 4/6 [00:33<00:17,  8.93s/it]
```

```
Done for alpha: 1000
```

```
 83%|████████  | 5/6 [01:15<00:18, 18.87s/it]
```

```
Done for alpha: 2000
```

```
100%|██████████| 6/6 [02:36<00:00, 37.39s/it]
```

In [52]:

```
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)
```

```
log_loss for c =  10 is 0.09505549957875219
log_loss for c =  50 is 0.07936086553025797
log_loss for c =  100 is 0.07865200494745483
log_loss for c =  500 is 0.07805499740565107
log_loss for c =  1000 is 0.0789473292463268
log_loss for c =  2000 is 0.07861410042716639
```

In [54]:

```
#AS we can see n_estimators = 500 gives best result.
#lets apply that and find the feature importances.
r_cfl=RandomForestClassifier(n_estimators=500,random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
#we have store the feature importances value in importance_of_feature
importance_of_feature = r_cfl.feature_importances_
```

In [56]:

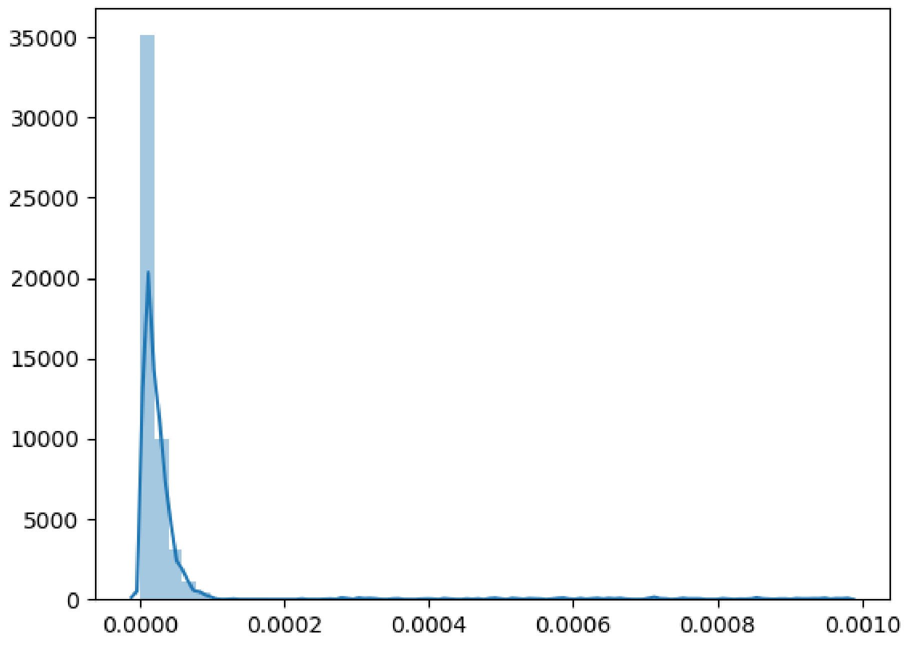
```
importance_of_feature
```

Out[56]:

```
array([0.00611162, 0.00596424, 0.00885048, ..., 0.        , 0.        ,
       0.        ])
```

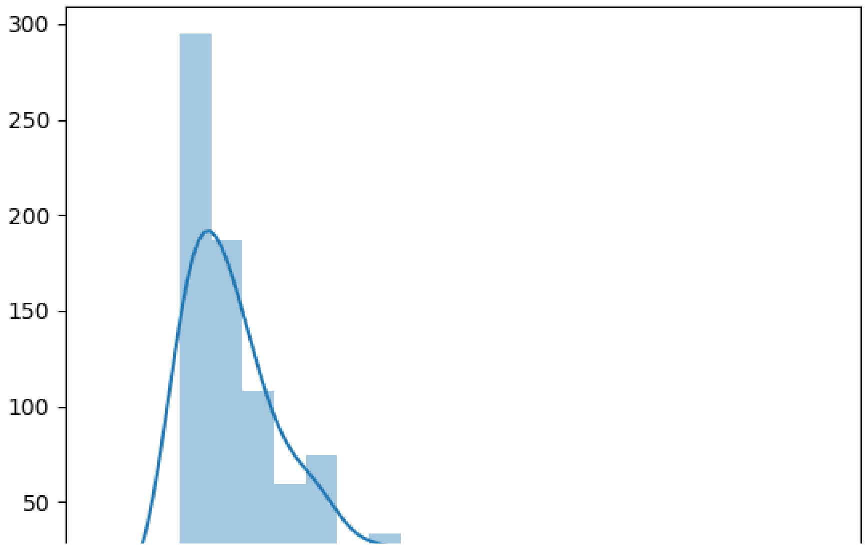
In [55]:

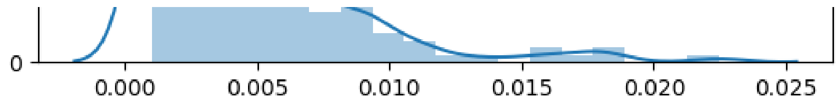
```
sns.distplot(importance_of_feature[importance_of_feature<0.001])
plt.show()
```



In [57]:

```
sns.distplot(importance_of_feature[importance_of_feature>0.001])
plt.show()
```





In [67]:

```
from sklearn.feature_selection import SelectFromModel
#we are setting the threshold 0.001 means only values greater than 0.001 are accepted.
sfm = SelectFromModel(r_cfl, threshold=0.001)
sfm.fit(X_train, y_train)
#fit has to be only on train data
```

Out[67]:

```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=True,
                                                 class_weight=None,
                                                 criterion='gini',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=50, n_jobs=-1,
                                                 oob_score=False,
                                                 random_state=42, verbose=0,
                                                 warm_start=False),
                max_features=None, norm_order=1, prefit=False, threshold=0.001)
```

In [68]:

```
X_important_train = sfm.transform(X_train)
X_important_cv = sfm.transform(X_cv)
X_important_test = sfm.transform(X_test)
```

In [ ]:

```
##Applying RandomForest
```

In [71]:

```
alpha=[10,50,100,500]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    print('Done for alpha:',i)
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_important_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_important_train,y_train)
    predict_y = sig_clf.predict_proba(X_important_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
```

```
Done for alpha: 10
```

```
 25%|██        | 1/4 [00:01<00:03,  1.32s/it]
```

```
Done for alpha: 50
```

```
 50%|█████     | 2/4 [00:03<00:03,  1.63s/it]
```

```
Done for alpha: 100
```

```
 75%|███████▌  | 3/4 [00:07<00:02,  2.19s/it]
```

```
Done for alpha: 500
```

```
100%|██████████| 4/4 [00:20<00:00,  5.44s/it]
```

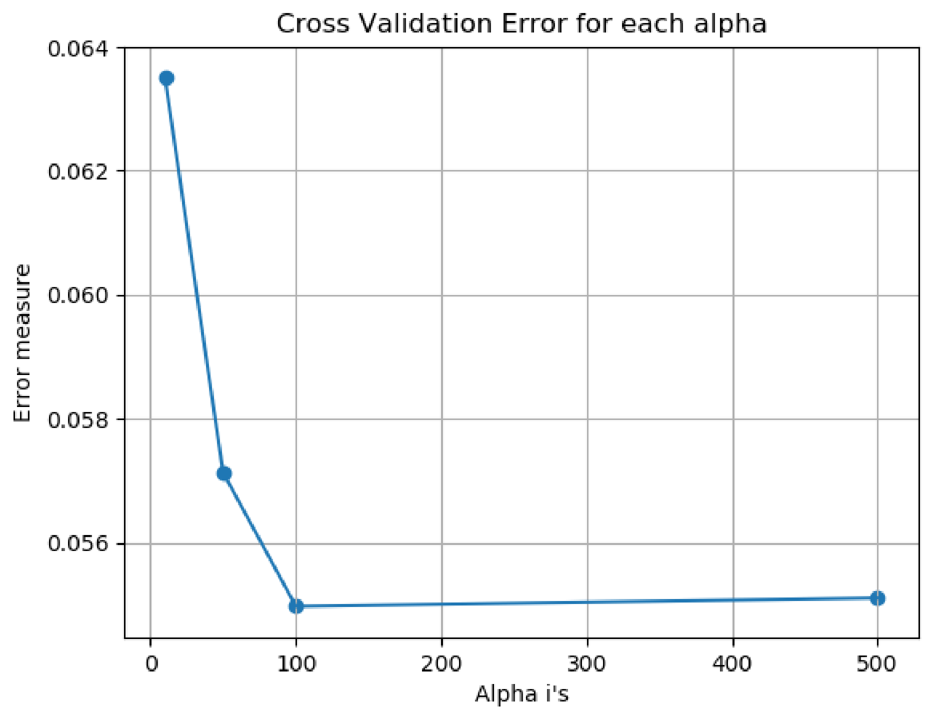
In [72]:

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

```
log_loss for c =  10 is 0.06350141101355673
log_loss for c =  50 is 0.0571349137463913
log_loss for c =  100 is 0.05497853688040346
log_loss for c =  500 is 0.055113253469029996
```

In [73]:

```
best_alpha = np.argmin(cv_log_error_array)

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



In [74]:

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

predict_y = sig_clf.predict_proba(X_important_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_important_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_important_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
```

```
For values of best alpha =  100 The train log loss is: 0.014988740452209453
For values of best alpha =  100 The cross validation log loss is: 0.05497853688040346
For values of best alpha =  100 The test log loss is: 0.03754687141581994
```

In [ ]:

```
##Applying XGBOOST
```

In [75]:

```
alpha=[10,50,100,200,500]
cv_log_error_array=[]
for i in tqdm(alpha):
    print('Done for alpha:',i)
    r_cfl=XGBClassifier(n_estimators=i,random_state=42,nthread=8)
    r_cfl.fit(X_important_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_important_train,y_train)
    predict_y = sig_clf.predict_proba(X_important_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))
```

```
  0%|          | 0/5 [00:00<?, ?it/s]
```

```
Done for alpha: 10
```

```
 20%|██        | 1/5 [00:03<00:12,  3.21s/it]
```

```
Done for alpha: 50
```

```
 40%|████      | 2/5 [00:16<00:19,  6.38s/it]
```

```
Done for alpha: 100
```

```
 60%|██████    | 3/5 [00:42<00:24, 12.07s/it]
```

```
Done for alpha: 200
```

```
 80%|████████  | 4/5 [01:23<00:20, 20.77s/it]
```

```
Done for alpha: 500
```

```
100%|██████████| 5/5 [02:36<00:00, 36.58s/it]
```

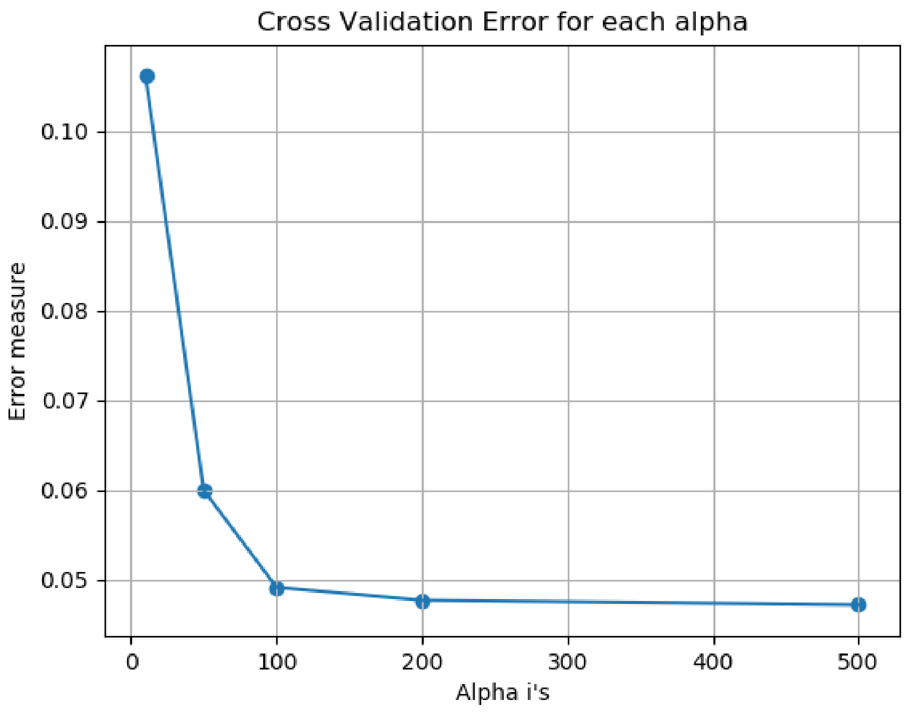
In [76]:

```
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)
```

```
log_loss for c =  10 is 0.10611306765414572
log_loss for c =  50 is 0.05996392104508787
log_loss for c =  100 is 0.04915055829496832
log_loss for c =  200 is 0.04771148299025181
log_loss for c =  500 is 0.047205560363089163
```

In [77]:

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



In [79]:

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

predict_y = sig_clf.predict_proba(X_important_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_important_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_important_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
```

```
For values of best alpha =  500 The train log loss is: 0.012261453583013523
For values of best alpha =  500 The cross validation log loss is: 0.047205560363089163
For values of best alpha =  500 The test log loss is: 0.0223644591259956
```

we managed to bring the test log loss to 0.022. However, let's add tetragrams to the final data as an additional feature and see if we can reduce the log loss less than 0.01

In [ ]:

```
#### Let's try the above using tetragrams
```

asm and byte unigrams + top1000_byte_bigrams+ top1000_opcodes_bi_trigrams+ top500 pixel_images+top1000_opcodes_bi_tetragrams

In [155]:

```
#along with tetragrams
final_data_2 = pd.concat([result_x,  top1000_byte_bigram_df,
top1000_opcodes_bi_tri_df,top1000_opcodes_bi_tetr_df, top500_image_df], axis = 1, join = 'inner')
```

In [156]:

```
final_data_2 = final_data_2.drop(['Unnamed: 0', 'Unnamed: 0.1', 'ID'],axis = 1)
```

In [81]:

```
X_2 = final_data_2
y = result_y
```

In [82]:

```
#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(X_2, y, stratify=y, test_size=0.20,
random_state=42)

#Split the train data into train and cross validation by maintaining same distribution of output v
araible '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,
random_state=42)
```

In [84]:

```
print(X_train.shape)
print(y_train.shape)
print(X_cv.shape)
print(y_cv.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(6955, 3807)
(6955, 1)
(1739, 3807)
(1739, 1)
(2174, 3807)
(2174, 1)
```

In [85]:

```
alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    print('Done for alpha:',i)
    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))
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

```
Done for alpha: 10
```

```
 17%|█         | 1/6 [00:02<00:13,  2.79s/it]
```

```
Done for alpha: 50
```

```
 33%|███       | 2/6 [00:07<00:13,  3.28s/it]
```

```
Done for alpha: 100
```

```
 50%|█████     | 3/6 [00:14<00:13,  4.45s/it]
```

```
Done for alpha: 500
```

```
 67%|██████▋   | 4/6 [00:38<00:20, 10.21s/it]
```

```
Done for alpha: 1000
```

```
 83%|████████▎ | 5/6 [01:23<00:20, 20.89s/it]
```

```
Done for alpha: 2000
```

```
100%|██████████| 6/6 [02:49<00:00, 40.20s/it]
```

In [86]:

```
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)
```

```
log_loss for c =  10 is 0.10684204025424095
log_loss for c =  50 is 0.08074936559066412
log_loss for c =  100 is 0.07856820893363156
log_loss for c =  500 is 0.0828729514731124
log_loss for c =  1000 is 0.08297196064759552
log_loss for c =  2000 is 0.08247798326622338
```

In [88]:

```
alpha[best_alpha]
```

Out[88]:

```
100
```

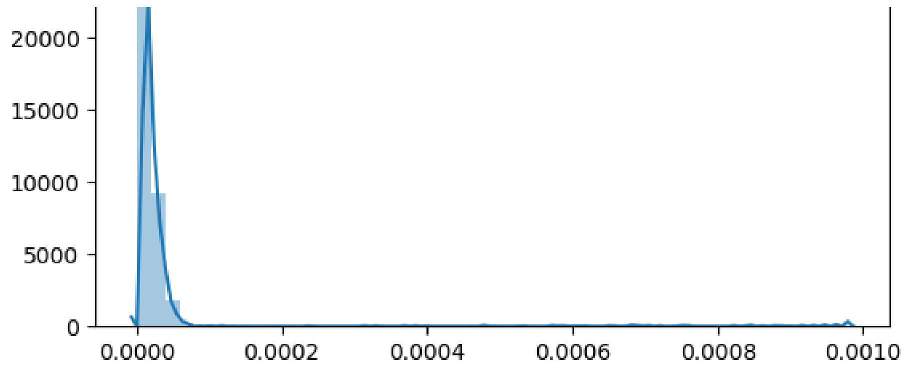
In [89]:

```
#AS we can see n_estimators = 100 gives best result.
#lets apply that and find the feature importances.
r_cfl=RandomForestClassifier(n_estimators=100,random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
#we have store the feature importances value in importance_of_feature
importance_of_feature = r_cfl.feature_importances_
```

In [90]:

```
sns.distplot(importance_of_feature[importance_of_feature<0.001])
plt.show()
```

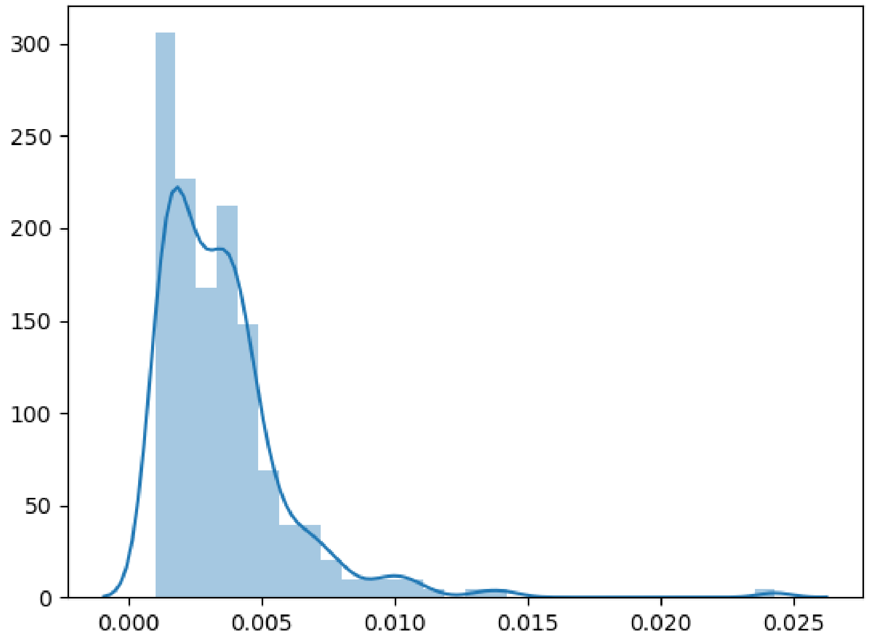
40000
35000
30000
25000



- first training with randomForest to find the best hyperparameter on the entire feature
- as we can more than 25000 features have feature importance almost 0.
- so take all the features which have feature importance more than 0.001

In [119]:

```
sns.distplot(importance_of_feature[importance_of_feature>0.001])
plt.show()
```



In [97]:

```
from sklearn.feature_selection import SelectFromModel
#we are setting the threshold 0.001 means only values greater than 0.001 are accepted.
sfm = SelectFromModel(r_cfl, threshold=0.001)
sfm.fit(X_train, y_train)
#fit has to be only on train data
```

Out[97]:

```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=True,
                                                 class_weight=None,
                                                 criterion='gini',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100, n_jobs=-1,
                                                 oob_score=False,
                                                 random_state=42, verbose=0,
                                                 warm_start=False),
                max_features=None, norm_order=1, prefit=False, threshold=0.001)
```

In [98]:

```
X_important_train = sfm.transform(X_train)
X_important_cv = sfm.transform(X_cv)
X_important_test = sfm.transform(X_test)
```

In [ ]:

```
## applying randomforest
```

In [99]:

```
alpha=[10,50,100,500]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    print('Done for alpha:',i)
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_important_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_important_train,y_train)
    predict_y = sig_clf.predict_proba(X_important_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
```

```
Done for alpha: 10
```

```
 25%|██        | 1/4 [00:01<00:04,  1.34s/it]
```

```
Done for alpha: 50
```

```
 50%|█████     | 2/4 [00:03<00:03,  1.67s/it]
```

```
Done for alpha: 100
```

```
 75%|███████▌  | 3/4 [00:07<00:02,  2.23s/it]
```

```
Done for alpha: 500
```

```
100%|██████████| 4/4 [00:20<00:00,  5.51s/it]
```

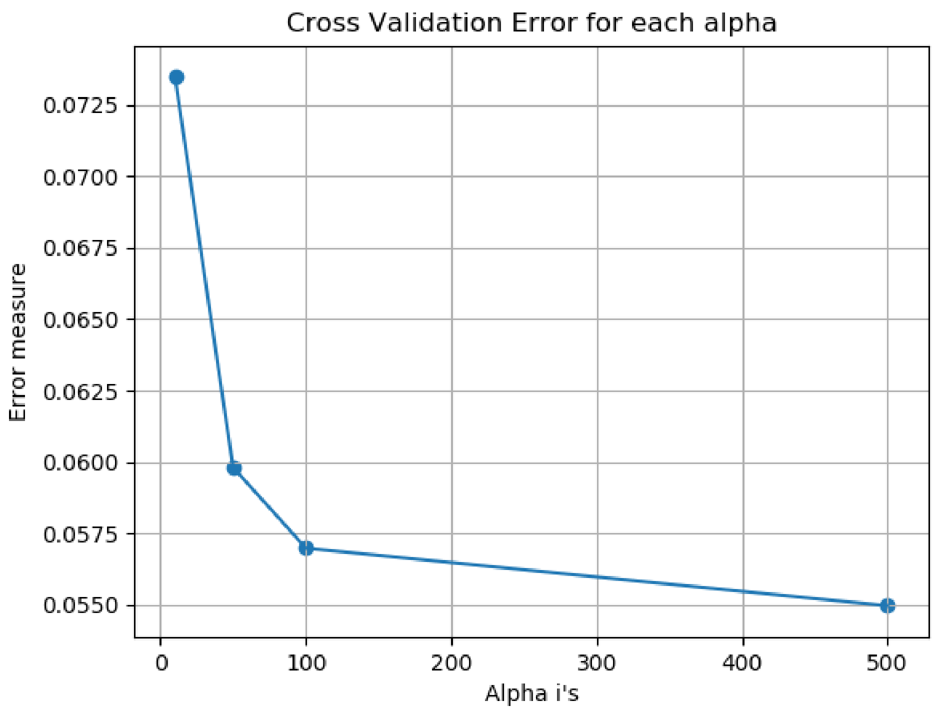
In [100]:

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

```
log_loss for c =  10 is 0.07347708383343933
log_loss for c =  50 is 0.05980378140840617
log_loss for c =  100 is 0.056978588788908116
log_loss for c =  500 is 0.05496017716369276
```

In [101]:

```
best_alpha = np.argmin(cv_log_error_array)

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



In [102]:

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

predict_y = sig_clf.predict_proba(X_important_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_important_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_important_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
```

```
For values of best alpha =  500 The train log loss is: 0.015097573256710876
For values of best alpha =  500 The cross validation log loss is: 0.05496017716369276
For values of best alpha =  500 The test log loss is: 0.038400979971908285
```

In [ ]:

```
## apply XGBOOST
```

In [103]:

```
alpha=[10,50,100,200,500]
cv_log_error_array=[]
for i in tqdm(alpha):
    print('Done for alpha:',i)
    r_cfl=XGBClassifier(n_estimators=i,random_state=42,nthread=8)
    r_cfl.fit(X_important_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_important_train,y_train)
    predict_y = sig_clf.predict_proba(x_important_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))
```

```
  0%|          | 0/5 [00:00<?, ?it/s]
```

```
Done for alpha: 10
```

```
 20%|██        | 1/5 [00:03<00:15,  3.79s/it]
```

```
Done for alpha: 50
```

```
 40%|████      | 2/5 [00:19<00:22,  7.36s/it]
```

```
Done for alpha: 100
```

```
 60%|██████    | 3/5 [00:48<00:27, 13.99s/it]
```

```
Done for alpha: 200
```

```
 80%|████████  | 4/5 [01:37<00:24, 24.32s/it]
```

```
Done for alpha: 500
```

```
100%|██████████| 5/5 [03:09<00:00, 44.69s/it]
```

In [104]:

```
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)
```

```
log_loss for c =  10 is 0.10680487113076424
log_loss for c =  50 is 0.06175004902230347
log_loss for c =  100 is 0.05064799526138072
log_loss for c =  200 is 0.04861710737050381
log_loss for c =  500 is 0.047560268241013
```

In [105]:

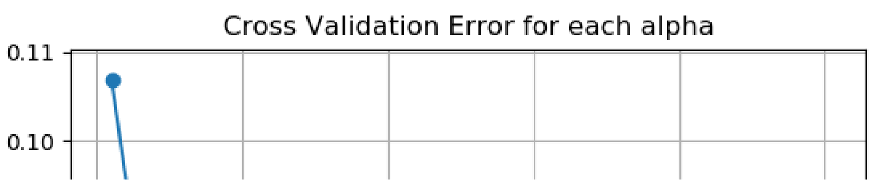
```
alpha[best_alpha]
```

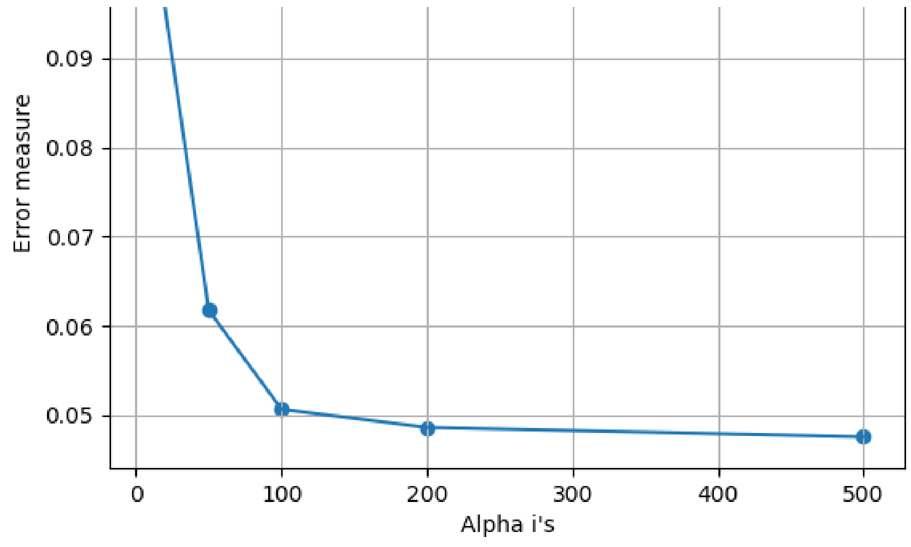
Out[105]:

```
500
```

In [106]:

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





In [107]:

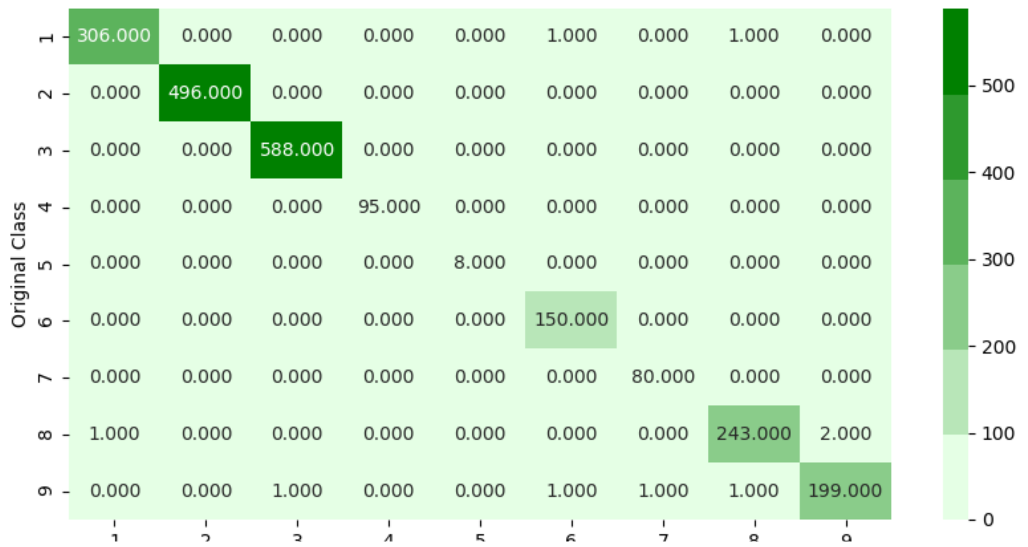
```
r_cfl=XGBClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_important_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_important_train, y_train)

predict_y = sig_clf.predict_proba(X_important_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_important_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_important_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
```

```
For values of best alpha =  500 The train log loss is: 0.012202078523820025
For values of best alpha =  500 The cross validation log loss is: 0.047560268241013
For values of best alpha =  500 The test log loss is: 0.02274749639838474
```

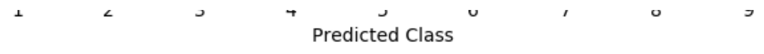
In [117]:

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

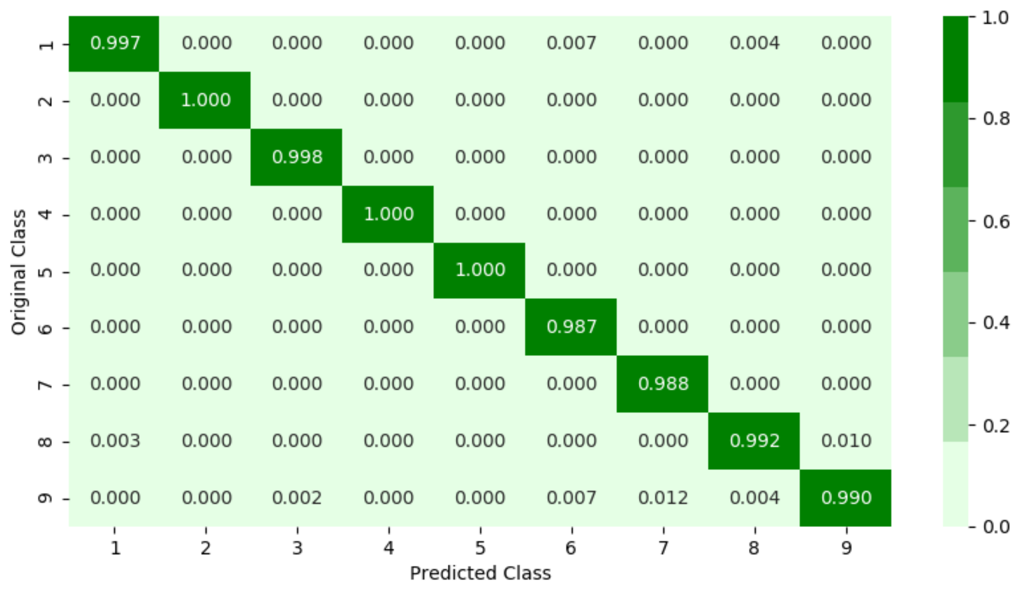
```
Number of misclassified points  0.41398344066237347
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```



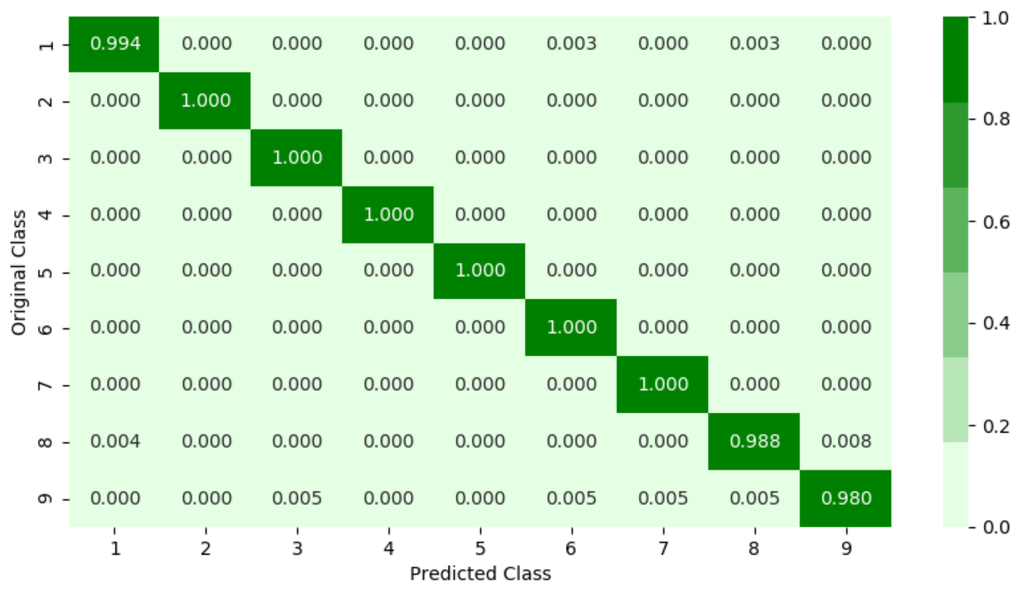
	1	2	3	4	5	6	7	8	9
1	306.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
2	0.000	496.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	95.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	8.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	150.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	80.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	0.000	243.000	2.000
9	0.000	0.000	1.000	0.000	0.000	1.000	1.000	1.000	199.000



```
-------------------------------------------------- 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 [167]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','log loss']
print("ft1 = asm and byte unigrams,top1000_byte_bigrams,top1000_opcodes_bi_trigrams,top500 pixel_i
mages")
print("ft2 = asm and byte unigrams,top1000_byte_bigrams,top1000_opcodes_bi_trigrams,top500 pixel_i
mages,top1000_opcodes_bi_tetragrams")

ptable.add_row(["xg boost","ft1","0.025"])
ptable.add_row(["random forest","reduced ft1 to 1500 using SVD","0.313"])
ptable.add_row(["xg boost","ft1(with reduced features using random forest and selectfrommodel)","0
.0223"])
ptable.add_row(["xg boost","ft2(with reduced features using random forest and selectfrommodel)","0
.0227"])



print(ptable)
```

```
ft1 = asm and byte unigrams,top1000_byte_bigrams,top1000_opcodes_bi_trigrams,top500 pixel_images
ft2 = asm and byte unigrams,top1000_byte_bigrams,top1000_opcodes_bi_trigrams,top500
pixel_images,top1000_opcodes_bi_tetragrams
+---------------+--------------------------------------------------------------------+----------+
|     Model     |                              Features                              | log loss |
+---------------+--------------------------------------------------------------------+----------+
|    xg boost   |                                ft1                                 |  0.025   |
| random forest |                   reduced ft1 to 1500 using SVD                    |  0.313   |
|    xg boost   | ft1(with reduced features using random forest and selectfrommodel) |  0.0223  |
|    xg boost   | ft2(with reduced features using random forest and selectfrommodel) |  0.0227  |
+---------------+--------------------------------------------------------------------+----------+
```

Summary:

- The least log loss obtained was 0.223
- xgboost takes a lot of time for even small number of estimators.
- Used trauncated SVD too reduce the dimension to 1500 in which 98% variance was preserved.
- But with reduced dim through trauncated svd model did not do well log loss increased from 0.02 which was before to 0.3. So it was not useful
- Then we selected the best features through RandomForest and selected about 200 features.
- With those(200) features we used random forest and xgboost
- The loss reduced to some extent but it didn't go beyond 0.2
- Extracting features took a lot of time.