In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

In [0]:
```
%cd drive/My Drive
```

```
[Errno 2] No such file or directory: 'drive/My Drive'
/content/drive/My Drive
```

In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import pickle
```

In [0]:
```
data_size_byte = pd.read_pickle('data_size_byte')
```

In [0]:
```
byte_features=pd.read_csv("results(1).csv")
print (byte_features.head())
```

```
                    ID       0     1     2     3     4     5     6     7  \
0  01azqd4InC7m9JpocGv5  601905  3905  2816  3832  3345  3242  3650  3201
1  01IsoiSMh5gxyDYTl4CB   39755  8337  7249  7186  8663  6844  8420  7589
2  01jsnpXSAlgw6aPeDxrU   93506  9542  2568  2438  8925  9330  9007  2342
3  01kcPWA9K2BOxQeS5Rju   21091  1213   726   817  1257   625   550   523
4  01SuzwMJEIXsK7A8dQbl   19764   710   302   433   559   410   262   249

      8  ...    f7    f8    f9    fa    fb    fc    fd     fe     ff     ??
0  2965  ...  2804  3687  3101  3211  3097  2758  3099   2759   5753   1824
1  9291  ...   451  6536   439   281   302  7639   518  17001  54902   8588
2  9107  ...  2325  2358  2242  2885  2863  2471  2786   2680  49144    468
3  1078  ...   478   873   485   462   516  1133   471    761   7998  13940
4   422  ...   847   947   350   209   239   653   221    242   2199   9008

[5 rows x 258 columns]
```

In [0]:
```
df = byte_features
import math
for column in df.loc[:, '0':'ff']:
   p = (df[column]/df[column].sum())
   h = p.apply(lambda x: np.log(x))
   df[column+'ent'] = - p * h
```

In [0]:
```
df.columns.get_loc("0ent")
```

Out[9]:
```
258
```

In [0]:
```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns[:258]:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value -
    return result1
result = normalize(df)
```

In [0]:
```
result.head()
```

Out[11]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262786	0.005425	0.001558	0.002056	0.002038	0.001828	0.002057
1	01IsoiSMh5gxyDYTl4CB	0.017332	0.011665	0.004024	0.003866	0.005294	0.003867	0.004746
2	01jsnpXSAlgw6aPeDxrU	0.040801	0.013361	0.001420	0.001304	0.005454	0.005274	0.005077
3	01kcPWA9K2BOxQeS5Rju	0.009182	0.001635	0.000395	0.000430	0.000760	0.000347	0.000309
4	01SuzwMJEIXsK7A8dQbl	0.008603	0.000926	0.000159	0.000223	0.000332	0.000225	0.000147

5 rows × 514 columns

In [0]:
```
result = pd.merge(result, data_size_byte,on='ID', how='left')
result.head()
```

Out[12]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262786	0.005425	0.001558	0.002056	0.002038	0.001828	0.002057
1	01IsoiSMh5gxyDYTl4CB	0.017332	0.011665	0.004024	0.003866	0.005294	0.003867	0.004746
2	01jsnpXSAlgw6aPeDxrU	0.040801	0.013361	0.001420	0.001304	0.005454	0.005274	0.005077
3	01kcPWA9K2BOxQeS5Rju	0.009182	0.001635	0.000395	0.000430	0.000760	0.000347	0.000309
4	01SuzwMJEIXsK7A8dQbl	0.008603	0.000926	0.000159	0.000223	0.000332	0.000225	0.000147

5 rows × 516 columns

In [0]:
```
max_value = result['size'].max()
min_value = result['size'].min()
result['size'] = (result['size'] - min_value) / (max_value - min_value)
```

In [0]:
```
#op.head()
```

Out[138]:

	(00, 00)	(00, 01)	(00, 02)	(00, 03)	(00, 04)	(00, 05)	(00, 06)	(00, 07)	(00, 08)	(00, 09)	...	(1E, B2)	(93, 20)	(0D, EB)	(89, 58)	(13 2D)
0	274425	1269	1029	1469	1227	1144	1437	1263	1174	1270	...	NaN	NaN	NaN	NaN	NaN
1	21075	752	73	48	175	12	10	11	42	9	...	NaN	NaN	NaN	NaN	NaN
2	16798	596	159	144	513	595	557	146	528	154	...	5.0	3.0	3.0	3.0	1.0
3	10417	225	61	69	114	40	25	22	63	15	...	1.0	2.0	2.0	3.0	1.0

4 rows × 65542 columns

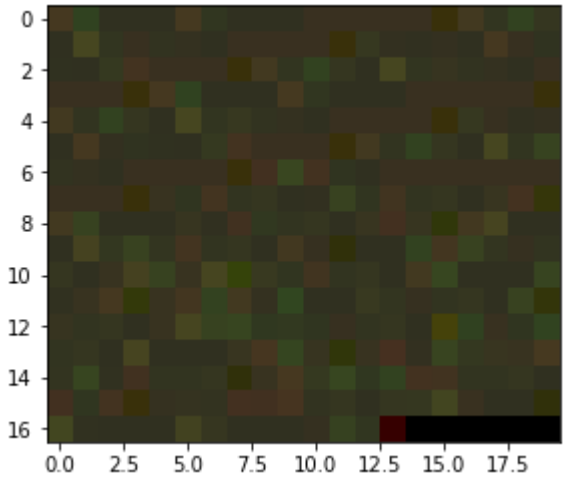
Extract byte Image feature

In [0]:
```
# https://github.com/dchad/malware-detection/blob/master/mmcc/feature-extraction.
import array
def read_image(filename):
    f = open(filename,'rb')
    ln = os.path.getsize(filename) # length of file in bytes
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
    a.fromfile(f,ln-rem)
    f.close()
    g = np.reshape(a,(len(a)//width,width))
    g = np.uint8(g).copy()
    g.resize((1000,))
    return list(g)
```

In [0]:
```
image_data = read_image("byteFiles/01azqd4InC7m9JpocGv5.txt")
```

In [0]:
```
%matplotlib inline
arr = np.array(image_data, dtype=np.uint8)
a = np.array(arr).reshape(-1,10).shape
#new_image = Image.fromarray(a)
im = np.array(image_data, dtype=np.uint8)
im.resize(17,20,3)

plt.imshow(im)
```

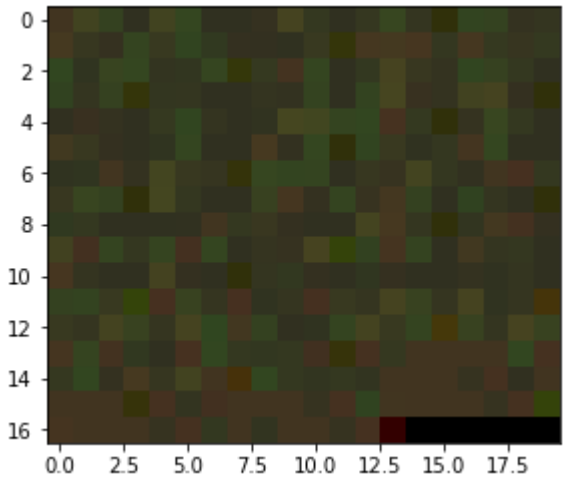
Out[8]:
```
<matplotlib.image.AxesImage at 0x7f4b42350668>
```



In [0]:
```
image_data = read_image("byteFiles/01SuzwMJEIXsK7A8dQbl.txt")
```

In [0]:
```
%matplotlib inline
arr = np.array(image_data, dtype=np.uint8)
a = np.array(arr).reshape(-1,10).shape
#new_image = Image.fromarray(a)
im = np.array(image_data, dtype=np.uint8)
im.resize(17,20,3)

plt.imshow(im)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x7f4b40b38400>
```



In [0]:
```
from collections import OrderedDict
files = ['byteFiles/'+ filename for filename in os.listdir('byteFiles')]
files_name = [os.path.splitext(filename)[0] for filename in os.listdir('byteFiles
lst = ['p'+str(i) for i in range(1000)]
lt = ['ID']
lst = str(lt+lst)[1:-1]
def file_add(count,file):
    byte_feature_file=open('byteimg.csv','w+')
    byte_feature_file.write(lst)
    byte_feature_file.write("\n")
    for i in range(len(count)):
        file_id = file[i].split("/")[1].split('.')[0]
        byte_feature_file.write(file_id+","+str(count[i])[1:-1])
        byte_feature_file.write("\n")

    byte_feature_file.close()

def count_words(filepath):
    image_data = read_image(filepath)
    return image_data
```

In [0]:
```
"""from __future__ import division, print_function
import os, glob, logging
from docopt import docopt
from multiprocessing import Pool
from functools import reduce

logging.basicConfig(level=logging.DEBUG)
#files = os.listdir('byteFiles')[:100]
#print(files)

if __name__ == '__main__':
   if not os.path.exists('byteFiles'):
      raise ValueError('Invalid data directory: %s' % 'byteFiles')
   num_processes = int(14)

   pool = Pool(processes=num_processes)
   per_doc_counts = pool.map(count_words, files)
   file_add(per_doc_counts,files)"""
```

Out[152]:
```
"from __future__ import division, print_function\nimport os, glob, logging\nfro
m docopt import docopt\nfrom multiprocessing import Pool\nfrom functools import
reduce\n\nlogging.basicConfig(level=logging.DEBUG)\n#files = os.listdir('byteFi
les')[:100]\n#print(files)\n\nif __name__ == '__main__':\n   if not os.path.exi
sts('byteFiles'):\n      raise ValueError('Invalid data directory: %s' % 'byteF
iles')\n   num_processes = int(14)\n\n   pool = Pool(processes=num_processes)\n
   per_doc_counts = pool.map(count_words, files)\n   file_add(per_doc_counts,fi
les)"
```

In [0]:
```
"""d = pd.read_csv("byteimg.csv")
#d['ID'] = d["'ID'"]
#d.drop("'ID'",axis=1,inplace=True)"""
```

In [0]:
```
"""d.columns = d.columns.str.replace(' ','')
d.rename(columns=lambda x: x[1:-1], inplace=True)
d.columns"""
```

Out[13]:
```
Index(['ID', 'p0', 'p1', 'p2', 'p3', 'p4', 'p5', 'p6', 'p7', 'p8',
       ...
       'p990', 'p991', 'p992', 'p993', 'p994', 'p995', 'p996', 'p997', 'p998',
       'p999'],
      dtype='object', length=1001)
```

In [0]:
```
"""result = pd.merge(d, result,on='ID', how='left')
result.head()"""
```

Out[14]:

	ID	p0	p1	p2	p3	p4	p5	p6	p7	p8	...	f8ent	f9ent	faen
0	01azqd4InC7m9JpocGv5	69	56	32	48	66	32	48	48	32	...	0.000990	0.001254	0.00117
1	01IsoiSMh5gxyDYTl4CB	67	55	32	48	49	32	50	52	32	...	0.001645	0.000217	0.00013
2	01jsnpXSAlgw6aPeDxrU	67	66	32	67	66	32	67	66	32	...	0.000664	0.000940	0.00107
3	01kcPWA9K2BOxQeS5Rju	54	65	32	70	70	32	54	56	32	...	0.000271	0.000237	0.00020
4	01SuzwMJEIXsK7A8dQbl	65	52	32	65	67	32	52	65	32	...	0.000292	0.000176	0.00010

5 rows × 1516 columns

In [0]:
```
"""df = result.copy()
df.drop('ID', axis=1, inplace=True)"""
```

In [0]:
```
"""df.head()"""
```

Out[27]:

	0	1	2	3	4	5	6	7	8	
0	0.262786	0.005425	0.001558	0.002056	0.002038	0.001828	0.002057	0.002945	0.002629	0.003
1	0.017332	0.011665	0.004024	0.003866	0.005294	0.003867	0.004746	0.006983	0.008258	0.000
2	0.040801	0.013361	0.001420	0.001304	0.005454	0.005274	0.005077	0.002154	0.008095	0.002
3	0.009182	0.001635	0.000395	0.000430	0.000760	0.000347	0.000309	0.000480	0.000950	0.000
4	0.008603	0.000926	0.000159	0.000223	0.000332	0.000225	0.000147	0.000228	0.000367	0.000

5 rows × 515 columns

In [0]:
```
"""import math
for column in df.loc[:, '0':'ff']:
   p = (df[column]/df[column].sum())
   h = p.apply(lambda x: np.log(x))
   df[column+'ent'] = - p * h"""
```

In [0]:
```
"""df_final = df.copy()
nunique = df_final.apply(pd.Series.nunique)
cols_to_drop = nunique[nunique == 1].index
df_final.drop(cols_to_drop, axis=1,inplace=True)
df_final.head()"""
```

Out[19]:

	p0	p1	p3	p4	p6	p7	p9	p10	p12	p13	...	f8ent	f9ent	faent	fbent	fc
0	69	56	48	66	48	48	48	48	48	48	...	0.000990	0.001254	0.001178	0.001390	0.000
1	67	55	48	49	50	52	48	52	53	67	...	0.001645	0.000217	0.000131	0.000172	0.001
2	67	66	67	66	67	66	67	66	48	48	...	0.000664	0.000940	0.001071	0.001297	0.000
3	54	65	70	70	54	56	65	51	49	54	...	0.000271	0.000237	0.000206	0.000279	0.000
4	65	52	65	67	52	65	48	48	65	67	...	0.000292	0.000176	0.000100	0.000139	0.000

5 rows × 1182 columns

In [0]:
```
#df_final.columns.get_loc("size")
```

In [0]:
```
"""# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns[667:925]:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value -
    return result1
result2 = normalize(df_final)"""
```

In [0]:
```
data_y = result['Class']
```

In [0]:
```
result.drop('Class',axis =1 , inplace=True)
```

In [0]:
```
result.head()
```

Out[18]:

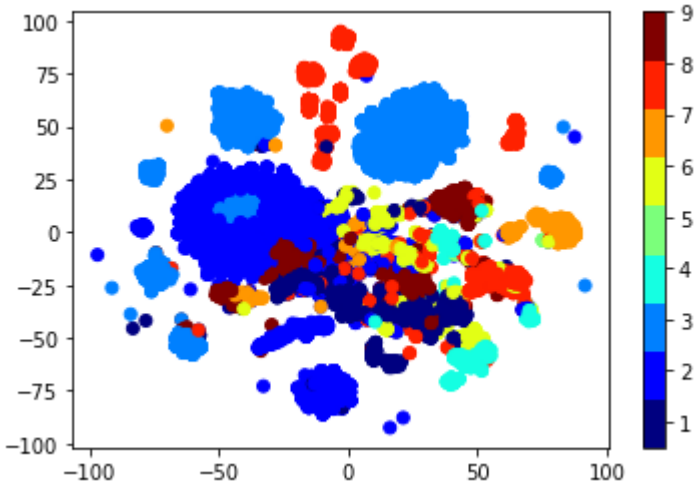
	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262786	0.005425	0.001558	0.002056	0.002038	0.001828	0.002057
1	01IsoiSMh5gxyDYTl4CB	0.017332	0.011665	0.004024	0.003866	0.005294	0.003867	0.004746
2	01jsnpXSAlgw6aPeDxrU	0.040801	0.013361	0.001420	0.001304	0.005454	0.005274	0.005077
3	01kcPWA9K2BOxQeS5Rju	0.009182	0.001635	0.000395	0.000430	0.000760	0.000347	0.000309
4	01SuzwMJEIXsK7A8dQbl	0.008603	0.000926	0.000159	0.000223	0.000332	0.000225	0.000147

5 rows × 515 columns

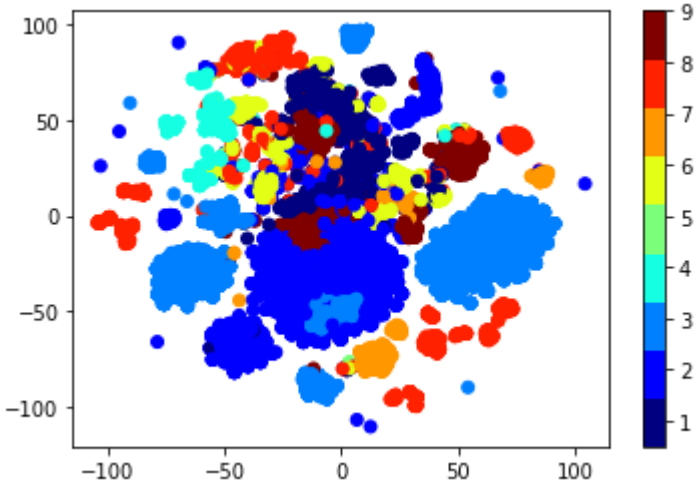
In [0]:
```
result.drop('ID',axis =1 , inplace=True)
```

3.2.4 Multivariate Analysis

In [0]:
```
%matplotlib inline
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



In [0]:
```
%matplotlib inline
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



Train Test split

In [0]:
```
#data_y = result['Class']
# split the data into test and train by maintaining same distribution of output v
X_train, X_test, y_train, y_test = train_test_split(result, data_y,stratify=data_
# split the train data into train and cross validation by maintaining same distri
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train
```

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

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

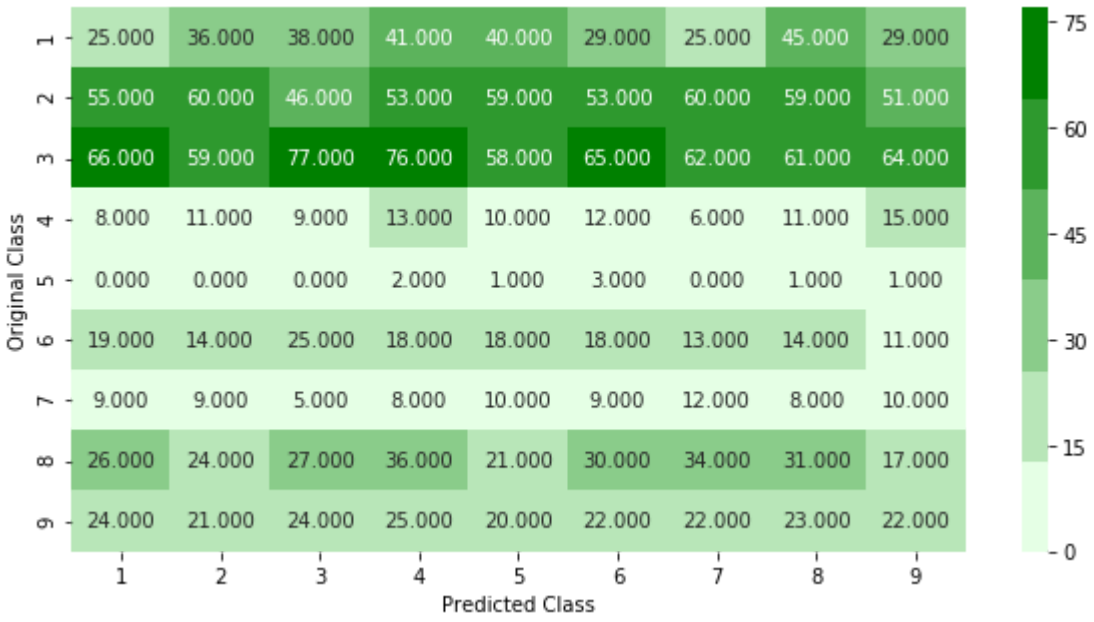
In [0]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are p

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

    # 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
    # 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 c
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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, yticklab
    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, yticklab
    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, yticklab
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

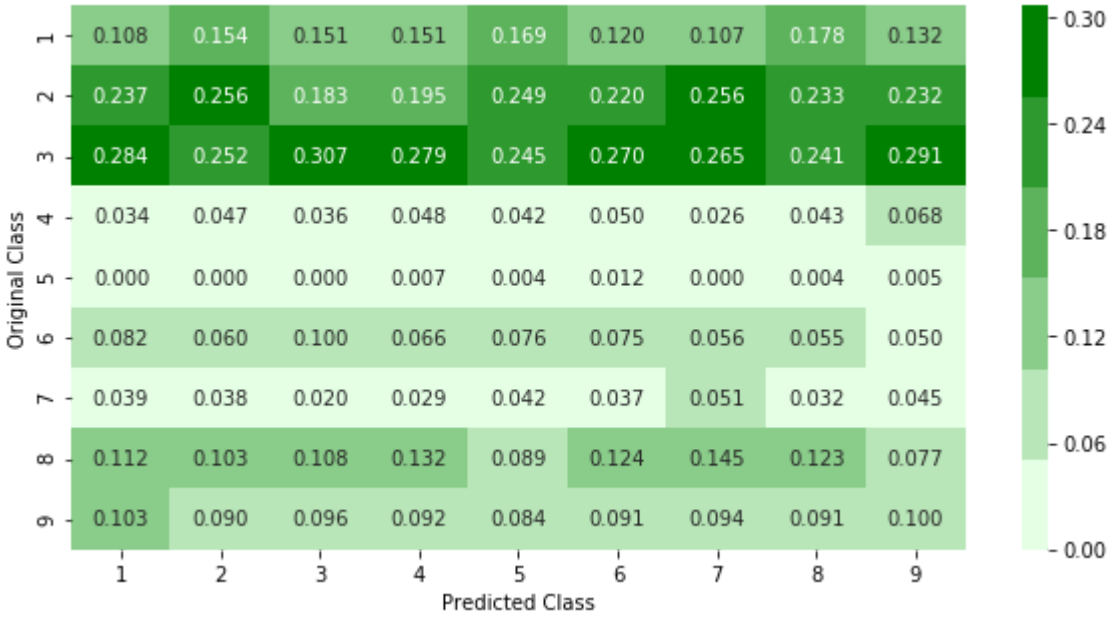
4.1.1. Random Model

In [0]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their s
# ref: https://stackoverflow.com/a/18662466/4084039

test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_pre


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y

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

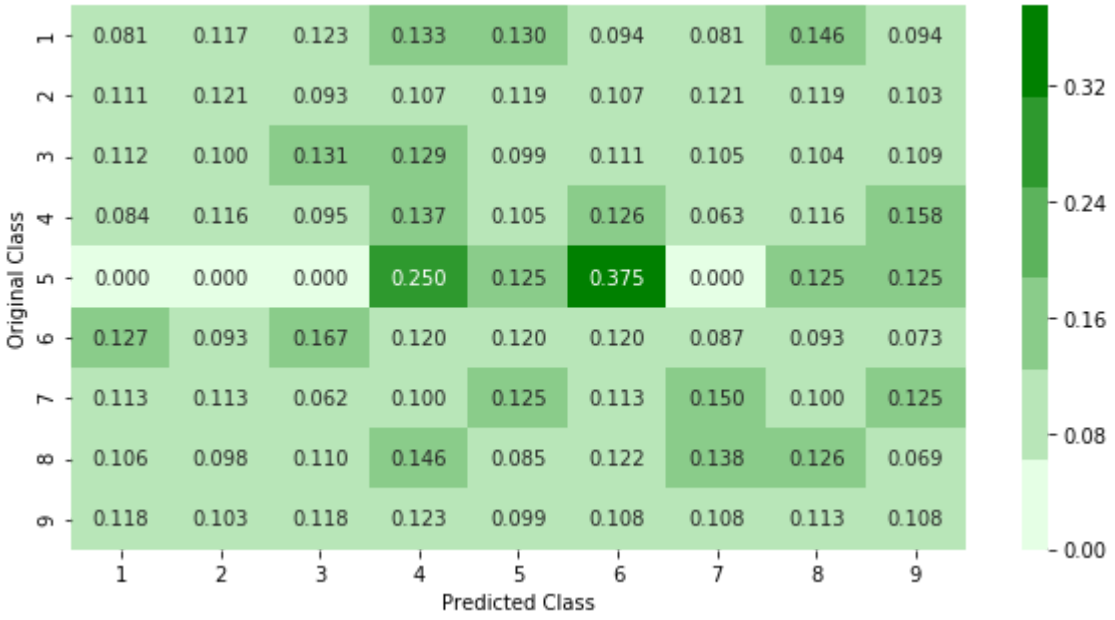
```
Log loss on Cross Validation Data using Random Model 2.441030573314521
Log loss on Test Data using Random Model 2.456064412523477
Number of misclassified points  88.08647654093836
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```




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




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



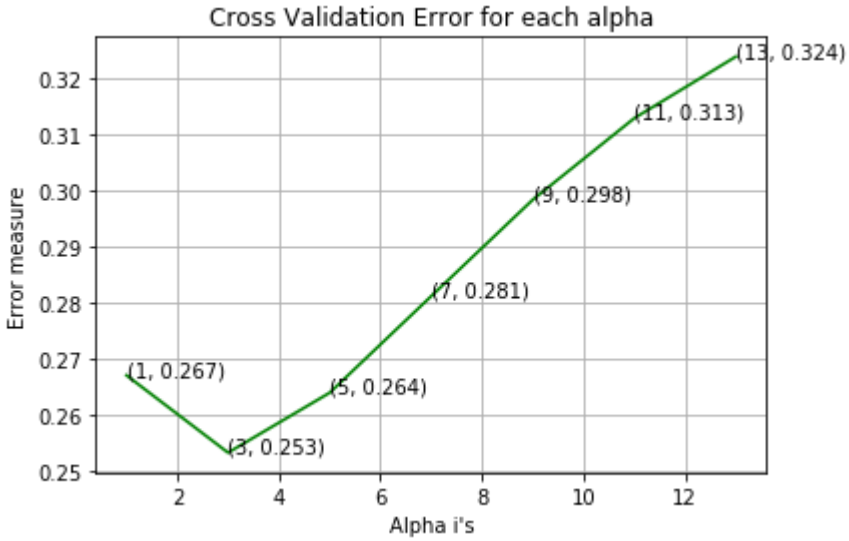

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.2. K Nearest Neighbour Classification

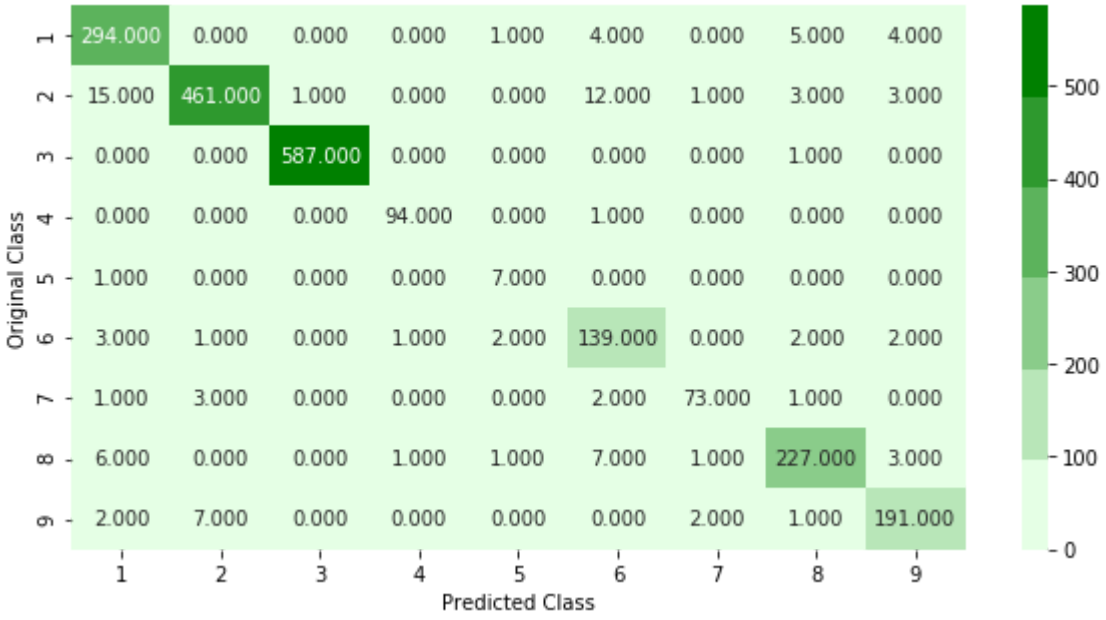
In [0]:
```
%matplotlib inline
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/les
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)  Posterior probabilities of classification
#-----------------------------------
# video link:
#-----------------------------------

alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, ep

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

best_alpha = np.argmin(cv_log_error_array)

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

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
```

```
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

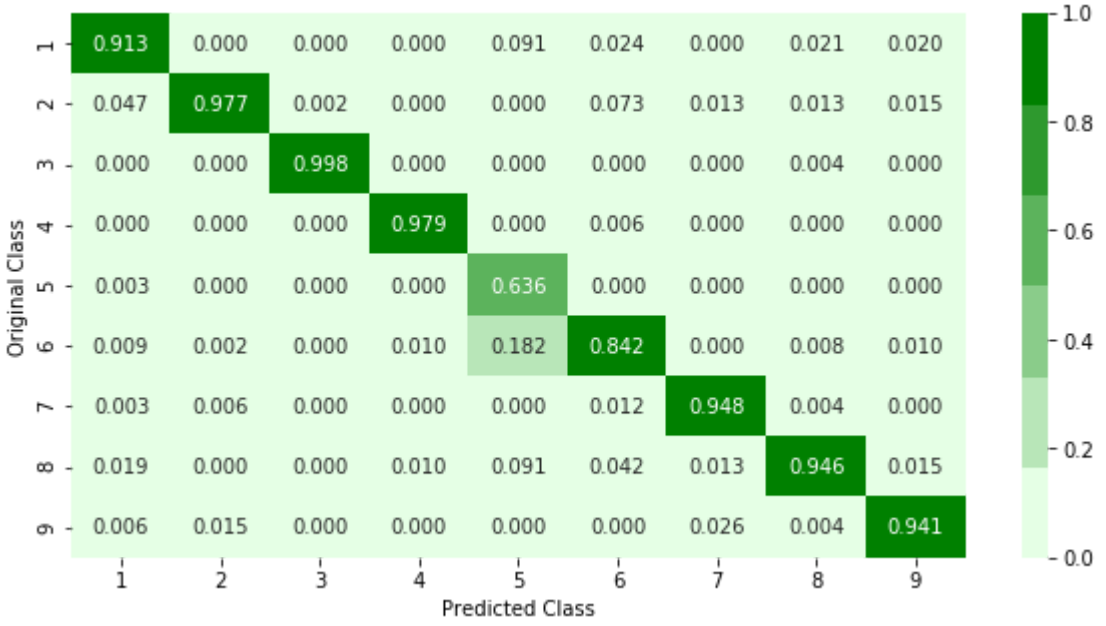
```
log_loss for k =  1 is 0.26709803945213145
log_loss for k =  3 is 0.2533192601082054
log_loss for k =  5 is 0.26398171199651554
log_loss for k =  7 is 0.2811875831364969
log_loss for k =  9 is 0.29837695103014916
log_loss for k =  11 is 0.3129065932312824
log_loss for k =  13 is 0.3238755407971689
```




```
For values of best alpha =  3 The train log loss is: 0.1289018092331439
For values of best alpha =  3 The cross validation log loss is: 0.2533192601082
054
For values of best alpha =  3 The test log loss is: 0.2036077024017526
Number of misclassified points  4.645814167433302
---------------------------------------------------- Confusion matrix -----------
---------------------------------------
```

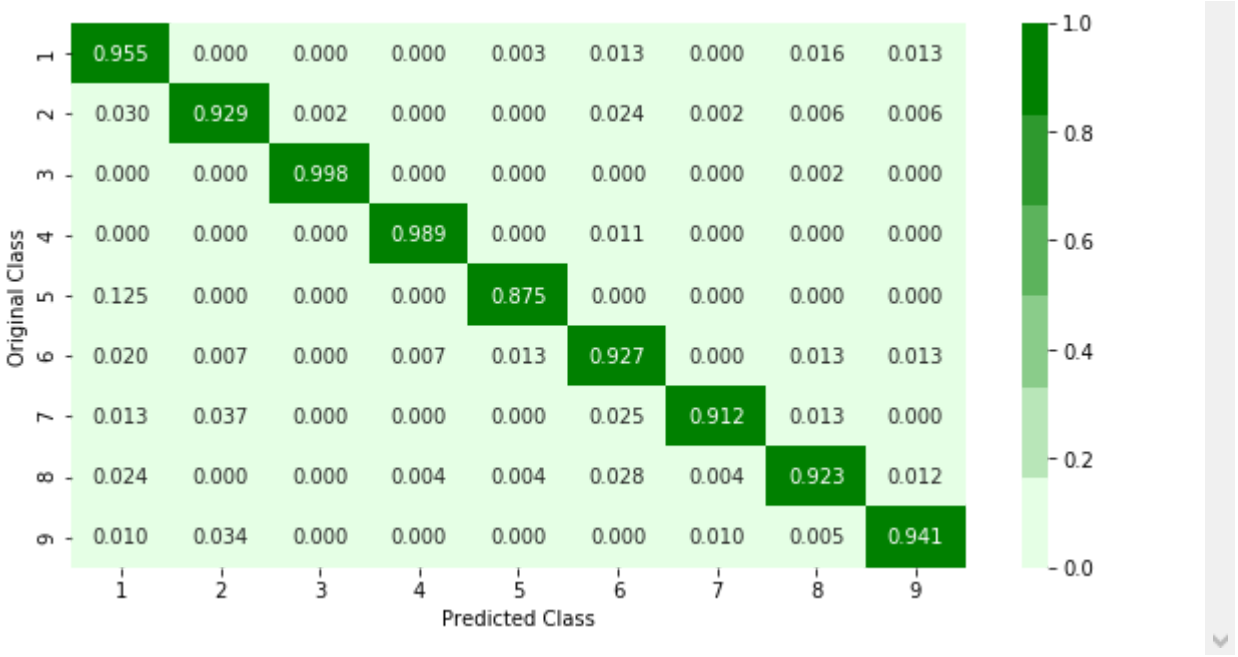


---------- Precision matrix ----------



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

---------- Recall matrix ----------



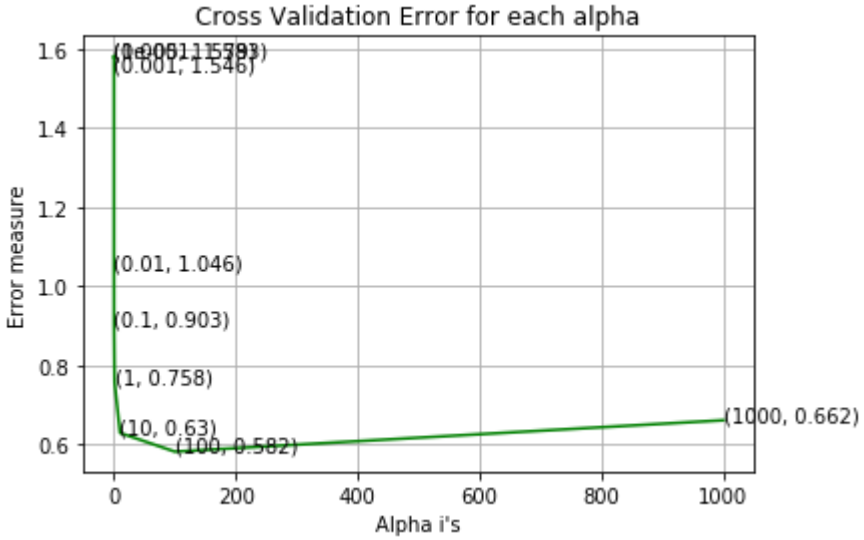

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.3. Logistic Regression

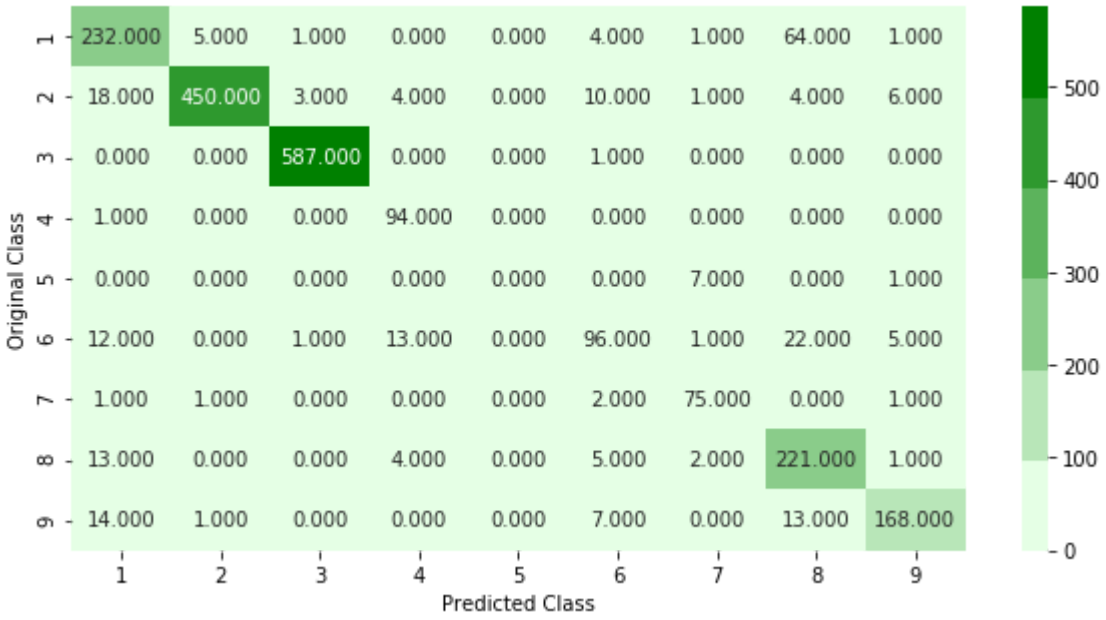
In [0]:
```
% matplotlib inline
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_

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='balan
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.cl
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.clas
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.5791581305422255
log_loss for c =  0.0001 is 1.5830716319332538
```

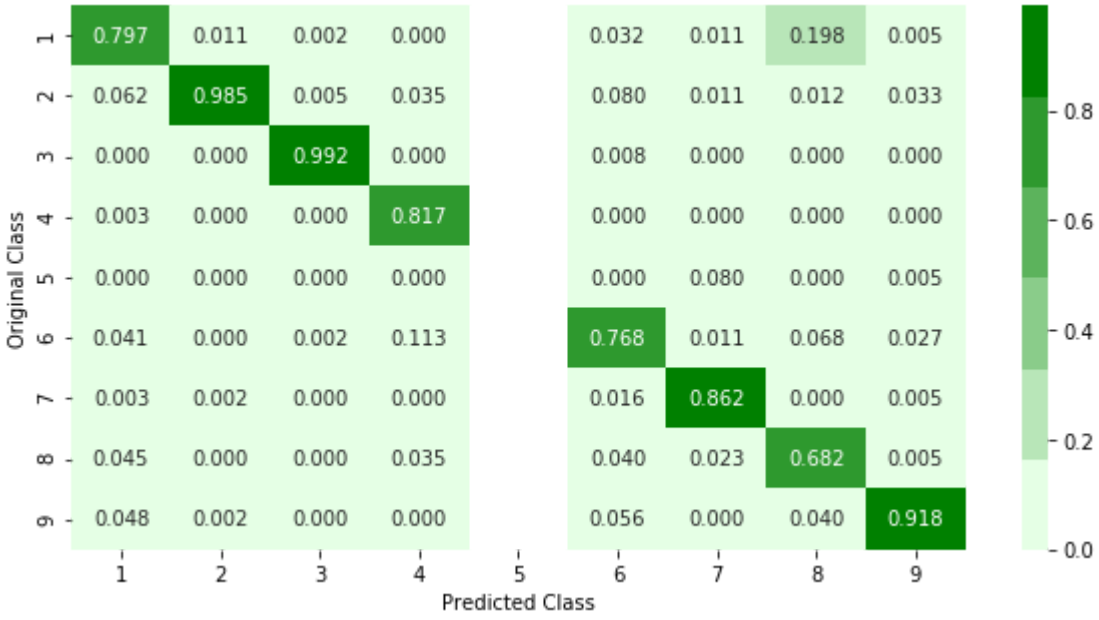
```
log_loss for c =  0.001 is 1.546222890831007
log_loss for c =  0.01 is 1.0464709557704475
log_loss for c =  0.1 is 0.9033650418738499
log_loss for c =  1 is 0.7576654608539423
log_loss for c =  10 is 0.6297816807138957
log_loss for c =  100 is 0.5824902249353602
log_loss for c =  1000 is 0.6617747430976415
```



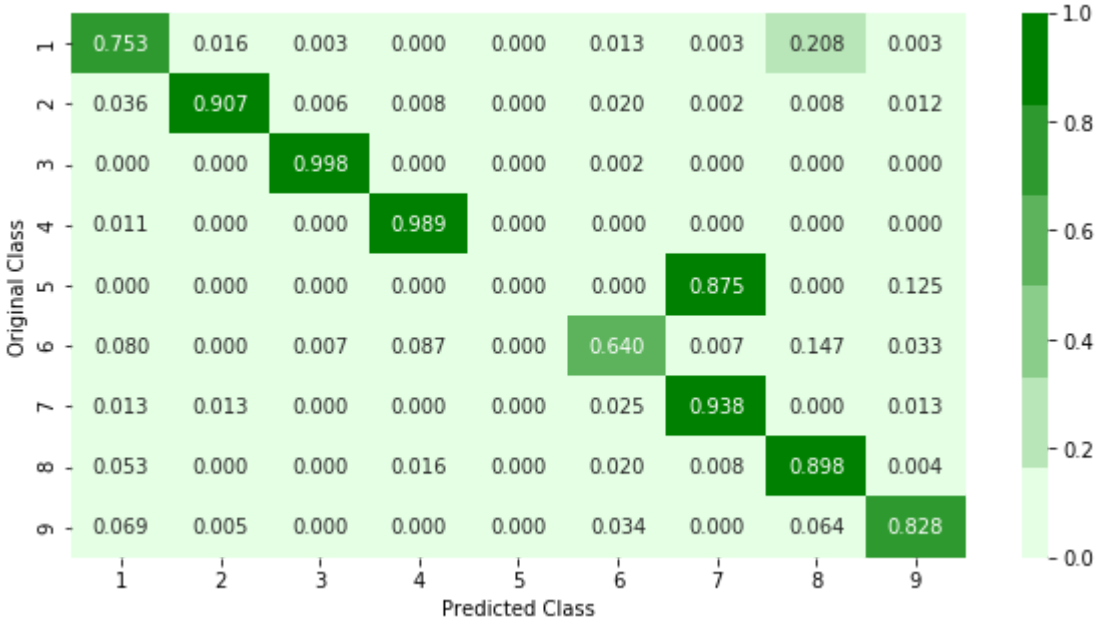

```
log loss for train data 0.48524200350942903
log loss for cv data 0.5824902249353602
log loss for test data 0.5117260489449236
Number of misclassified points  11.545538178472862
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```




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




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



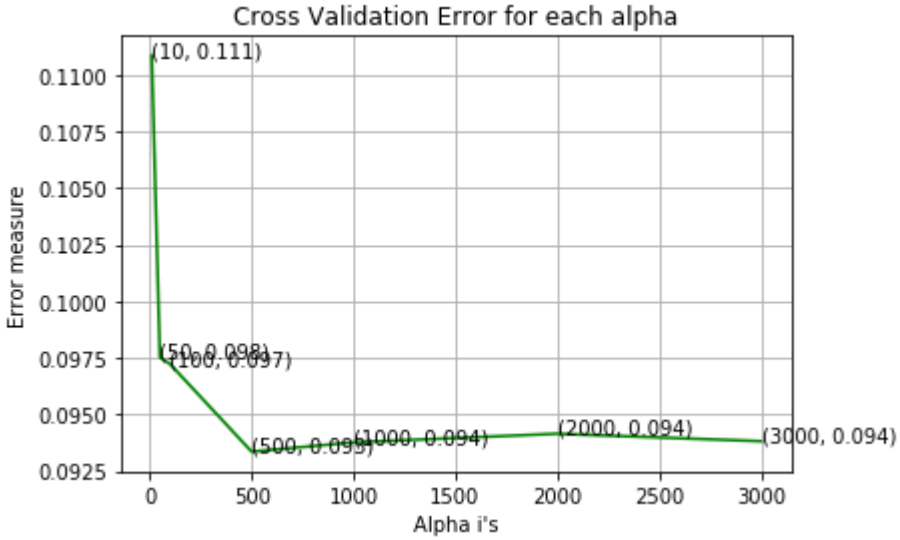

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.4. Random Forest Classifier

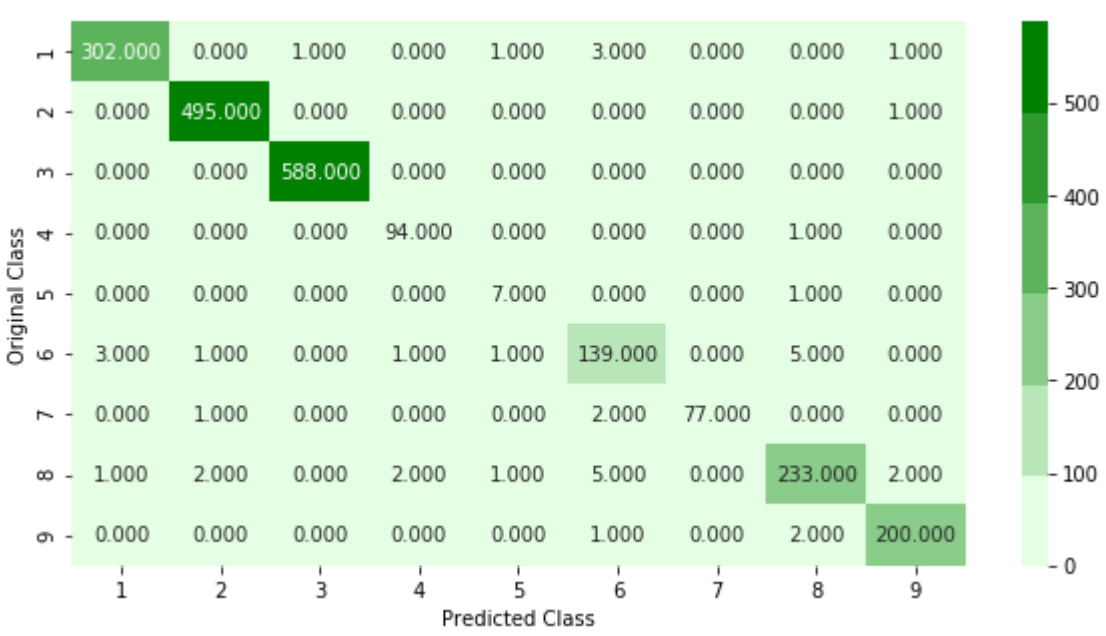
In [0]:
```
%matplotlib inline
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state
# class_weight=None)

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

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

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

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_, ep

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

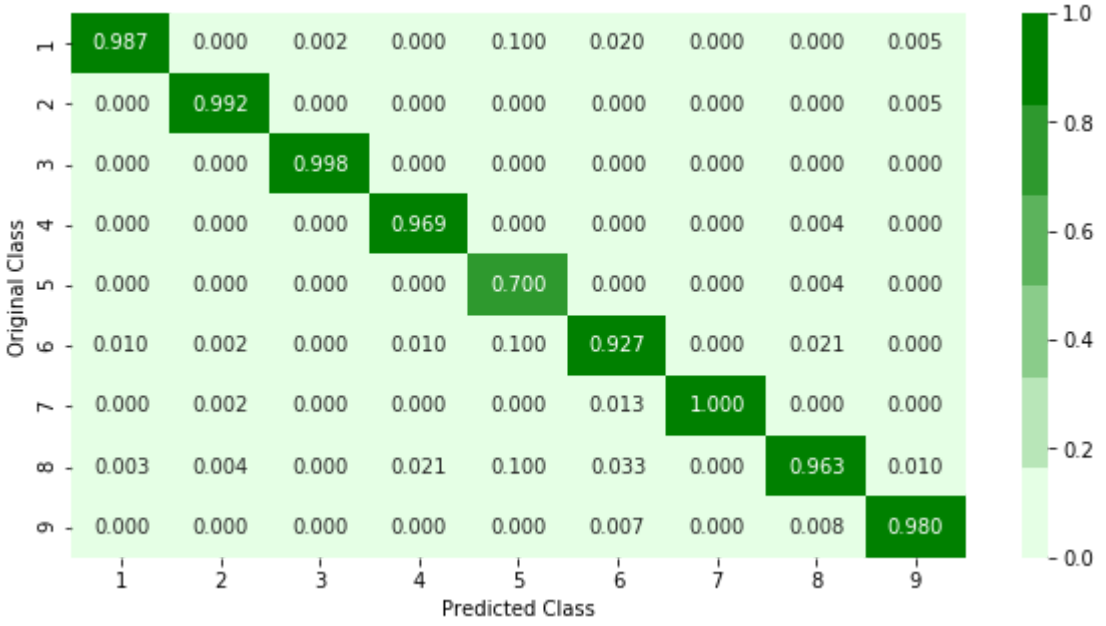
```
log_loss for c =  10 is 0.11085848284238148
log_loss for c =  50 is 0.0975264262040836
log_loss for c =  100 is 0.09722694054796645
log_loss for c =  500 is 0.09335822819848473
log_loss for c =  1000 is 0.093768971908842
log_loss for c =  2000 is 0.09414977118989808
log_loss for c =  3000 is 0.0938193962236563
```




```
For values of best alpha =  500 The train log loss is: 0.03242932816264531
For values of best alpha =  500 The cross validation log loss is: 0.09335822819
848473
For values of best alpha =  500 The test log loss is: 0.08153871966671211
Number of misclassified points  1.7939282428702852
---------------------------------------------------- Confusion matrix -----------
---------------------------------------
```

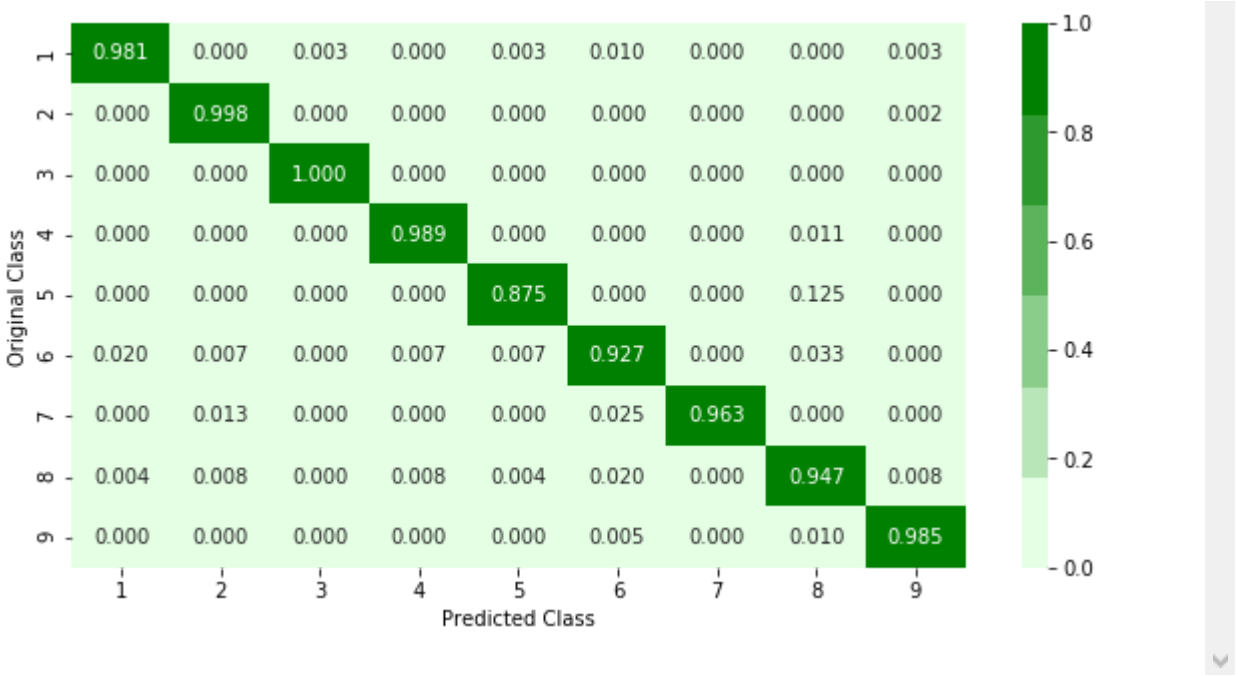


---------- Precision matrix ----------



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

---------- Recall matrix ----------



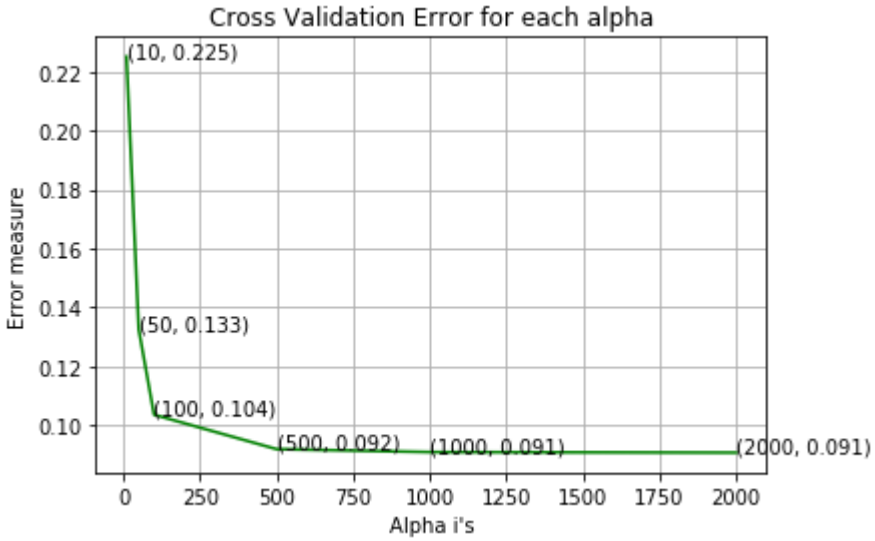

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.5. XgBoost Classification

In [0]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/la
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, s
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alp
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_r
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: Th
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/les
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/les
# -----------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

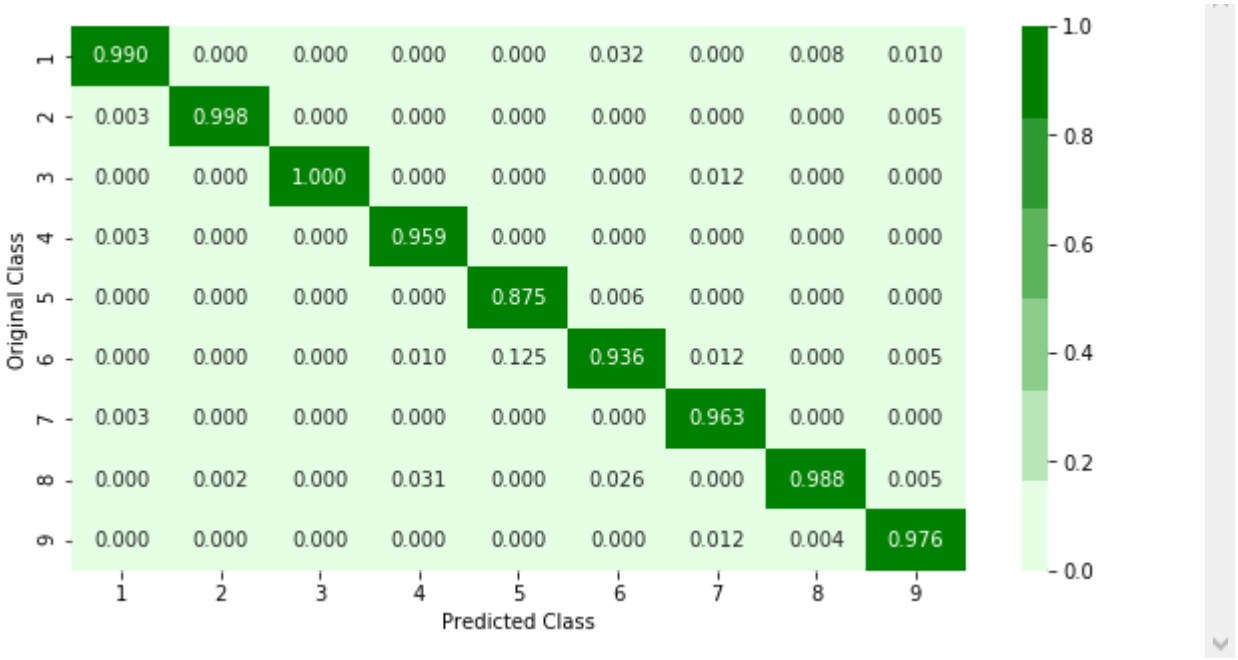
```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.22508599796019452
log_loss for c =  50 is 0.13253626132090074
log_loss for c =  100 is 0.10356652494418223
log_loss for c =  500 is 0.09188743351946499
log_loss for c =  1000 is 0.09088809415725702
log_loss for c =  2000 is 0.09069048063248059
```

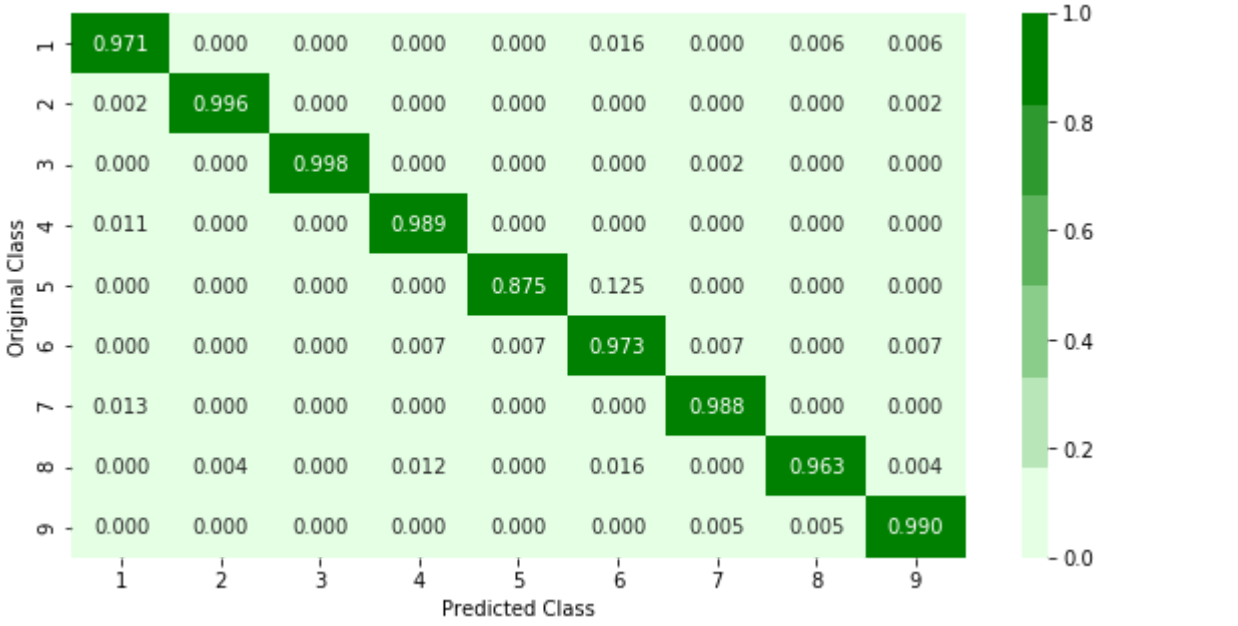



```
For values of best alpha =  2000 The train log loss is: 0.025172302056717756
For values of best alpha =  2000 The cross validation log loss is: 0.0906904806
3248059
For values of best alpha =  2000 The test log loss is: 0.06955789027897305
Number of misclassified points  1.3799448022079117
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```


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




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




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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [0]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgb
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=
random_cfl1.fit(X_train,y_train)
```

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

[Parallel(n_jobs=10)]: Using backend LokyBackend with 10 concurrent workers.
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/
process_executor.py:706: UserWarning: A worker stopped while some jobs were giv
en to the executor. This can be caused by a too short worker timeout or by a me
mory leak.
  "timeout or by a memory leak.", UserWarning
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/
process_executor.py:706: UserWarning: A worker stopped while some jobs were giv
en to the executor. This can be caused by a too short worker timeout or by a me
mory leak.
  "timeout or by a memory leak.", UserWarning
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/
process_executor.py:706: UserWarning: A worker stopped while some jobs were giv
en to the executor. This can be caused by a too short worker timeout or by a me
mory leak.
  "timeout or by a memory leak.", UserWarning
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/
process_executor.py:706: UserWarning: A worker stopped while some jobs were giv
en to the executor. This can be caused by a too short worker timeout or by a me
mory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=10)]: Done   5 tasks      | elapsed: 14.6min
[Parallel(n_jobs=10)]: Done  15 out of  30 | elapsed: 83.7min remaining: 83.7mi
n
[Parallel(n_jobs=10)]: Done  19 out of  30 | elapsed: 98.5min remaining: 57.1mi
n
[Parallel(n_jobs=10)]: Done  23 out of  30 | elapsed: 114.0min remaining: 34.7m
in
[Parallel(n_jobs=10)]: Done  27 out of  30 | elapsed: 120.2min remaining: 13.4m
in
[Parallel(n_jobs=10)]: Done  30 out of  30 | elapsed: 137.0min finished
```

Out[25]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=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_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=10,
          param_distributions={'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], 'co
lsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [0]:
```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 10, 'learning_rate': 0.03,
'colsample_bytree': 0.3}
```

In [0]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/la
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, s
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alp
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_r
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: Th
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/les
# ------------------------

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, ma
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

In [0]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/la
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, s
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alp
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_r
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: Th
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/les
# -----------------------

x_cfl=XGBClassifier(subsample=1, n_estimators=2000, learning_rate=0.03, colsample
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.02358091431675165
cv loss 0.0892635409842546
test loss 0.0923896596814937
```

In [1]:
```
from prettytable import PrettyTable
import sys
sys.stdout.write("\033[1;30m")

x = PrettyTable()
x.field_names = ["Model", "Train Loss","CV Loss", "Test Loss"]

x.add_row(["Random Model", '-', 2.4410, 2.4560])
x.add_row(["KNN", 0.1289, 0.2533, 0.2036])
x.add_row(["LR", 0.4852, 0.5824, 0.5117])
x.add_row(["RF", 0.0324, 0.0933, 0.0815])
x.add_row(["XGBoost", 0.0251, 0.0906, 0.0695])
x.add_row(["XGBoost(Random Search)", 0.0235, 0.0892, 0.0923])
print(x)
```

```
+-----------------------+------------+---------+-----------+
|         Model         | Train Loss | CV Loss | Test Loss |
+-----------------------+------------+---------+-----------+
|      Random Model     |     -      |  2.441  |   2.456   |
|          KNN          |   0.1289   |  0.2533 |   0.2036  |
|           LR          |   0.4852   |  0.5824 |   0.5117  |
|           RF          |   0.0324   |  0.0933 |   0.0815  |
|        XGBoost        |   0.0251   |  0.0906 |   0.0695  |
| XGBoost(Random Search) |   0.0235   |  0.0892 |   0.0923  |
+-----------------------+------------+---------+-----------+
```

STEPS TAKEN

1) Calculated entropy for 256 columns (Opcodes).

2) Image features and bigram features performed badly so I skipped it.

3) Trained various model using these features.

In [0]: