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
```

```
/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]:
```
asm_features=pd.read_csv("try_results.csv")
print(asm_features.head())
```

```
                    ID  .BSS:  .CODE  .Pav:  .bss:   .data:  .dll  .edata:  \
0  01azqd4InC7m9JpocGv5      0      0      0      0  1366755     0        0
1  01IsoiSMh5gxyDYTl4CB      0      0      0      0    24618     0        0
2  01jsnpXSAlgw6aPeDxrU      0      0      0      0      662     0        0
3  01kcPWA9K2BOxQeS5Rju      0      0      0      0       58     0        0
4  01SuzwMJEIXsK7A8dQbl      0      0      0     96     4686     0        0

   .idata:  .rdata:  ...  retn  rol  ror  rtn  shl  shr  std::  sub  xchg  xor
0     1158     2263  ...   290    0    0    0  158    2      0  648     0  418
1      616    26760  ...   252    0    0    0   23   14      0  216     0  106
2      304     1236  ...    27    0    0    0    0    0      0   91     0  199
3      127      381  ...    16    0    0    0    1    0      0    5     0   18
4      206        0  ...    33    0    2    0   32    0      0  363     9   19

[5 rows x 52 columns]
```

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

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

Out[6]:

	ID	.BSS:	.CODE	.Pav:	.bss:	.data:	.dll	.edata:	.idata:	.rdata:	...
0	01azqd4InC7m9JpocGv5	0	0	0	0	1366755	0	0	1158	2263	...
1	01IsoiSMh5gxyDYTl4CB	0	0	0	0	24618	0	0	616	26760	...
2	01jsnpXSAlgw6aPeDxrU	0	0	0	0	662	0	0	304	1236	...
3	01kcPWA9K2BOxQeS5Rju	0	0	0	0	58	0	0	127	381	...
4	01SuzwMJEIXsK7A8dQbl	0	0	0	96	4686	0	0	206	0	...

5 rows × 54 columns

In [0]:
```
result = result.loc[:, (result != 0).any(axis=0)]
```

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

Out[9]:

	ID	.Pav:	.bss:	.data:	.edata:	.idata:	.rdata:	.reloc:	.rsrc:	.text:	.
0	01azqd4InC7m9JpocGv5	0	0	1366755	0	1158	2263	0	0	23226	.
1	01IsoiSMh5gxyDYTl4CB	0	0	24618	0	616	26760	0	0	110032	.
2	01jsnpXSAlgw6aPeDxrU	0	0	662	0	304	1236	0	0	68915	.
3	01kcPWA9K2BOxQeS5Rju	0	0	58	0	127	381	3	3	782	.
4	01SuzwMJEIXsK7A8dQbl	0	96	4686	0	206	0	0	3	10456	.

5 rows × 48 columns

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

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

Out[13]:

	ID	.Pav:	.bss:	.data:	.edata:	.idata:	.rdata:	.reloc:	.rsrc:	.text:	.
0	01azqd4InC7m9JpocGv5	0	0	1366755	0	1158	2263	0	0	23226	.
1	01IsoiSMh5gxyDYTl4CB	0	0	24618	0	616	26760	0	0	110032	.
2	01jsnpXSAlgw6aPeDxrU	0	0	662	0	304	1236	0	0	68915	.
3	01kcPWA9K2BOxQeS5Rju	0	0	58	0	127	381	3	3	782	.
4	01SuzwMJEIXsK7A8dQbl	0	96	4686	0	206	0	0	3	10456	.

5 rows × 93 columns

In [0]:
```
df = df.fillna(0)
```

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

Out[17]:
```
92
```

In [0]:
```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns[:92]:
        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[19]:

	ID	.Pav:	.bss:	.data:	.edata:	.idata:	.rdata:	.reloc:	.r
0	01azqd4InC7m9JpocGv5	0.0	0.000000	0.542821	0.0	0.006936	0.000589	0.000000	0.000
1	01IsoiSMh5gxyDYTl4CB	0.0	0.000000	0.009777	0.0	0.003689	0.006969	0.000000	0.000
2	01jsnpXSAlgw6aPeDxrU	0.0	0.000000	0.000263	0.0	0.001821	0.000322	0.000000	0.000
3	01kcPWA9K2BOxQeS5Rju	0.0	0.000000	0.000023	0.0	0.000761	0.000099	0.001001	0.000
4	01SuzwMJEIXsK7A8dQbl	0.0	0.013393	0.001861	0.0	0.001234	0.000000	0.000000	0.000

5 rows × 93 columns

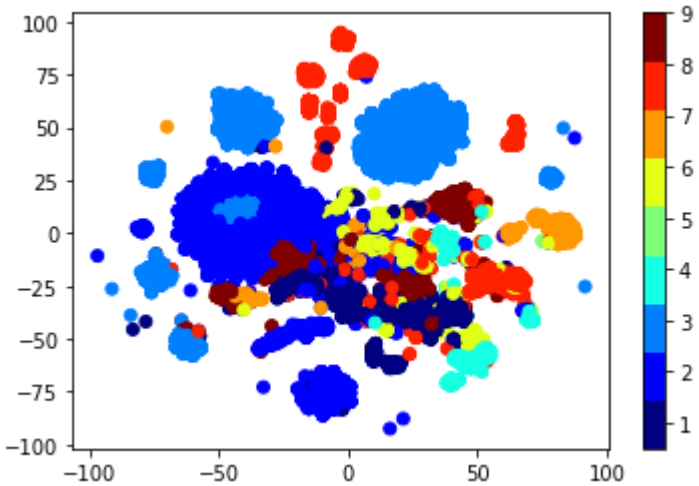
In [0]:
```
data_y = result['Class']
```

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

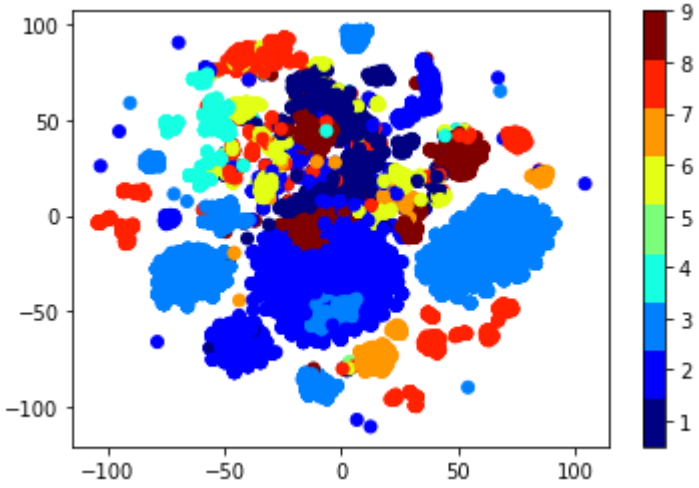
In [0]:
```
result.drop('ID',axis =1 , inplace=True)
```

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)
```

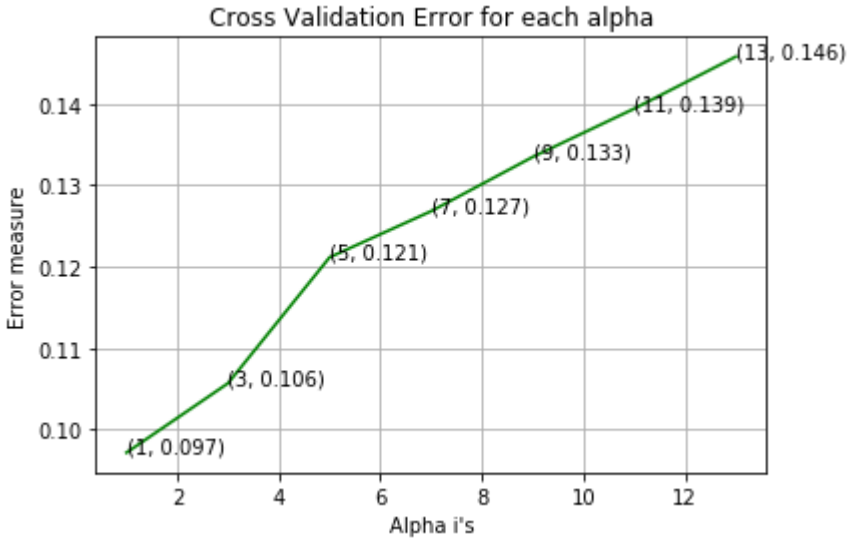
```
Log loss on Cross Validation Data using Random Model 2.481096531869642
Log loss on Test Data using Random Model 2.4807637525243433
Number of misclassified points  89.60441582336706
---------------------------------------------------- Confusion matrix -----------
--------------------------------------

<IPython.core.display.Javascript object>


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

<IPython.core.display.Javascript object>

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

<IPython.core.display.Javascript object>


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])
```

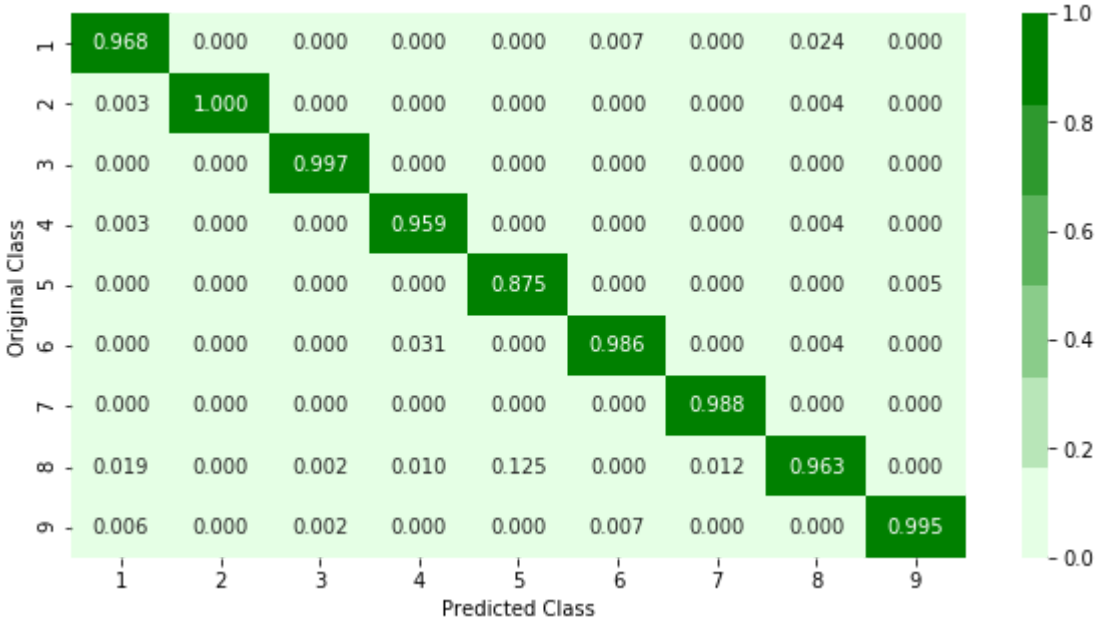
```
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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 0.0971514354217617
log_loss for k =  3 is 0.10567134407770849
log_loss for k =  5 is 0.12113113718958225
log_loss for k =  7 is 0.12679720511430087
log_loss for k =  9 is 0.1334531166397167
log_loss for k =  11 is 0.13938609743616942
log_loss for k =  13 is 0.14576873942346993
```

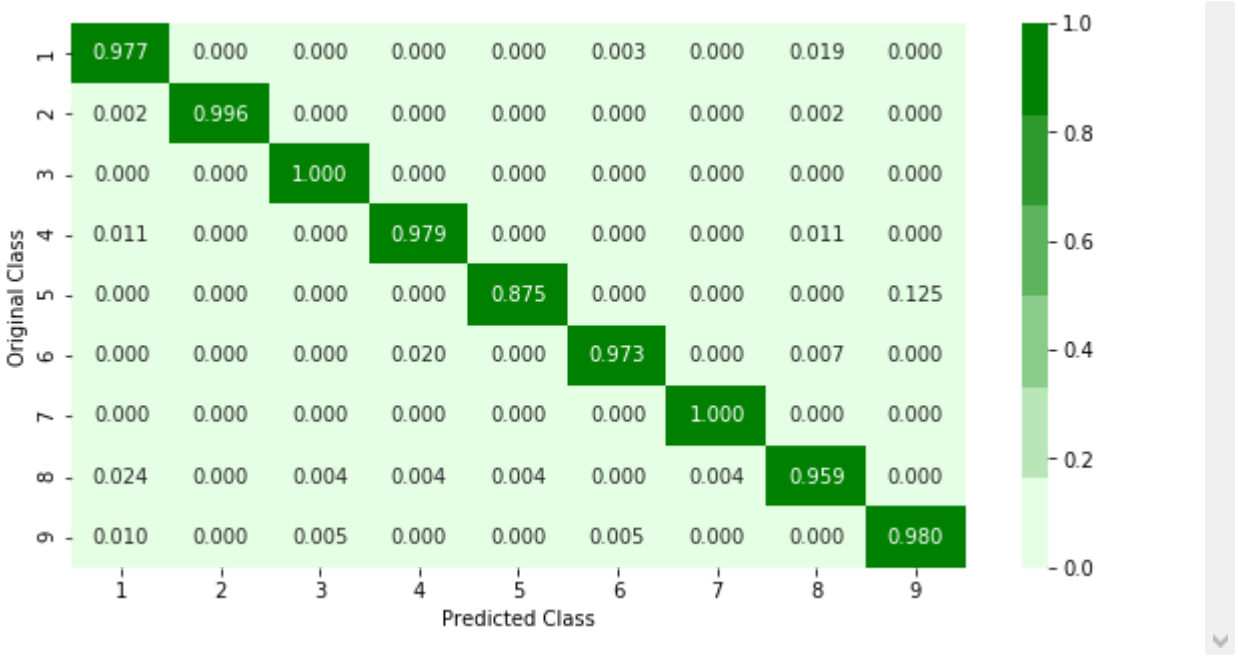



```
For values of best alpha =  1 The train log loss is: 0.025466872044967146
For values of best alpha =  1 The cross validation log loss is: 0.0971514354217
617
For values of best alpha =  1 The test log loss is: 0.08381273725005002
Number of misclassified points  1.3799448022079117
---------------------------------------------------- Confusion matrix -----------
---------------------------------------
```

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



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



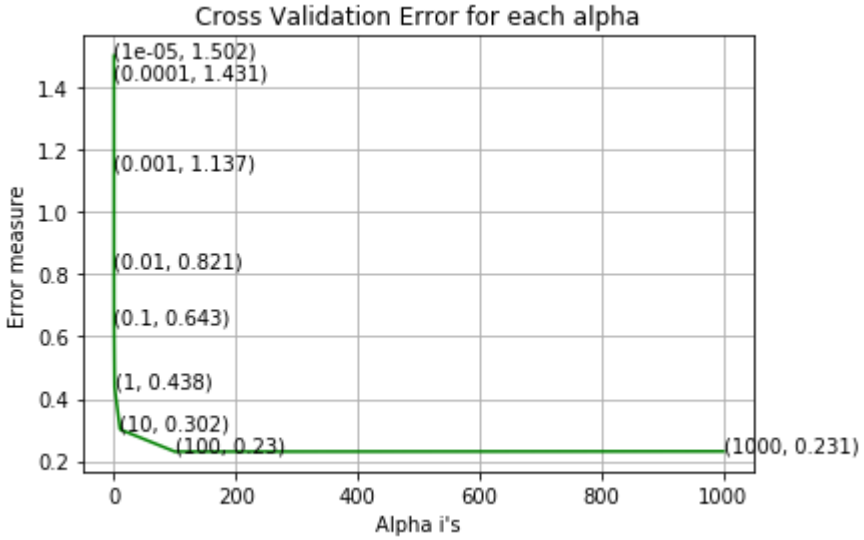

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.3. Logistic Regression

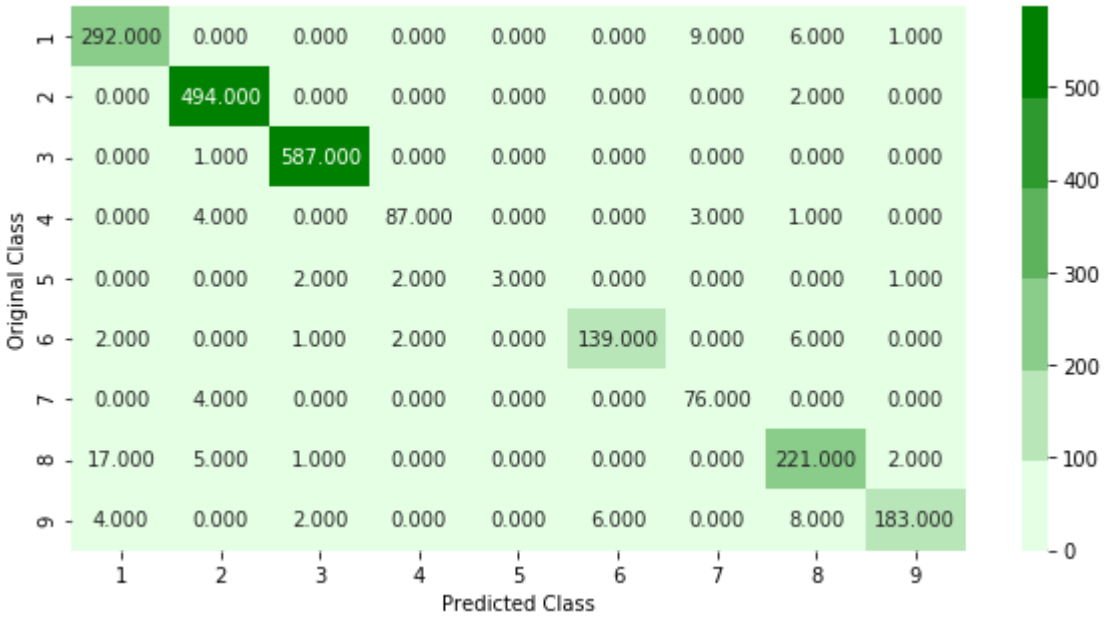
In [0]:
```
% matplotlib inline
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, 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 1.502478186761017
log_loss for c =  0.0001 is 1.431135835526333
```

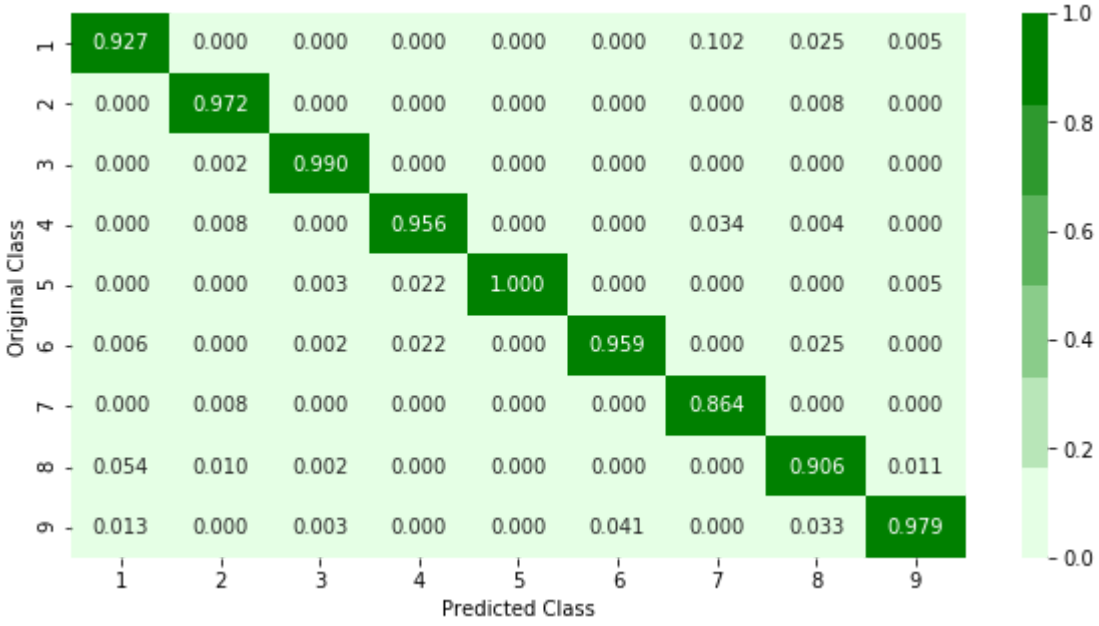
```
log_loss for c =  0.001 is 1.136680351515169
log_loss for c =  0.01 is 0.8212735649743487
log_loss for c =  0.1 is 0.6429126520596781
log_loss for c =  1 is 0.438053949482081
log_loss for c =  10 is 0.30179258469077536
log_loss for c =  100 is 0.230336238613876
log_loss for c =  1000 is 0.23139829699678915
```



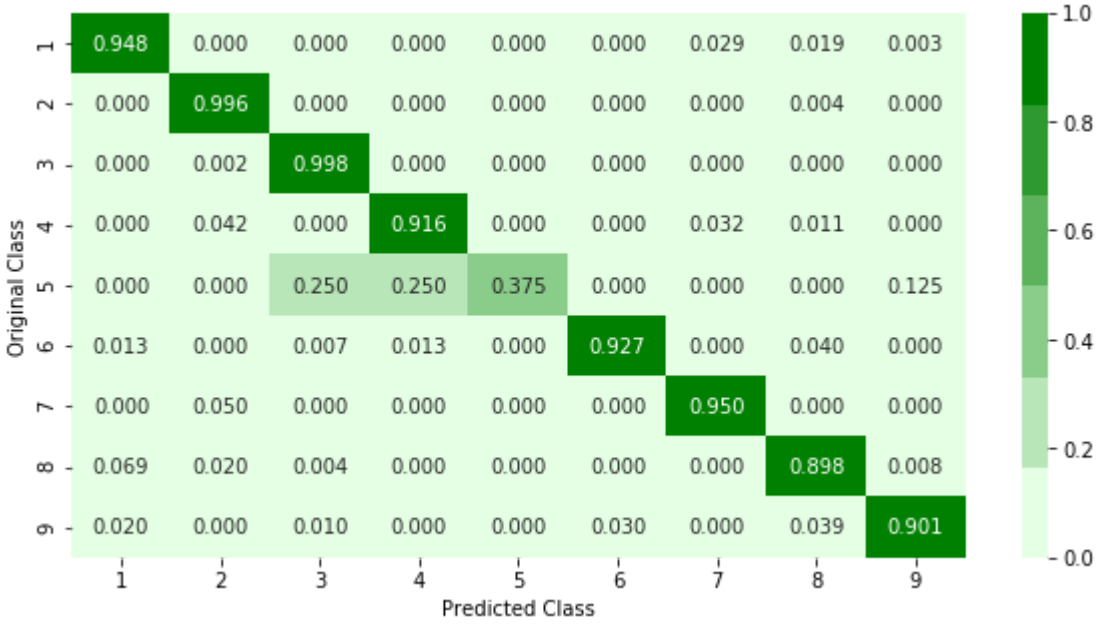

```
log loss for train data 0.20343106404471367
log loss for cv data 0.230336238613876
log loss for test data 0.19979007417357159
Number of misclassified points  4.231830726770929
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```




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




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



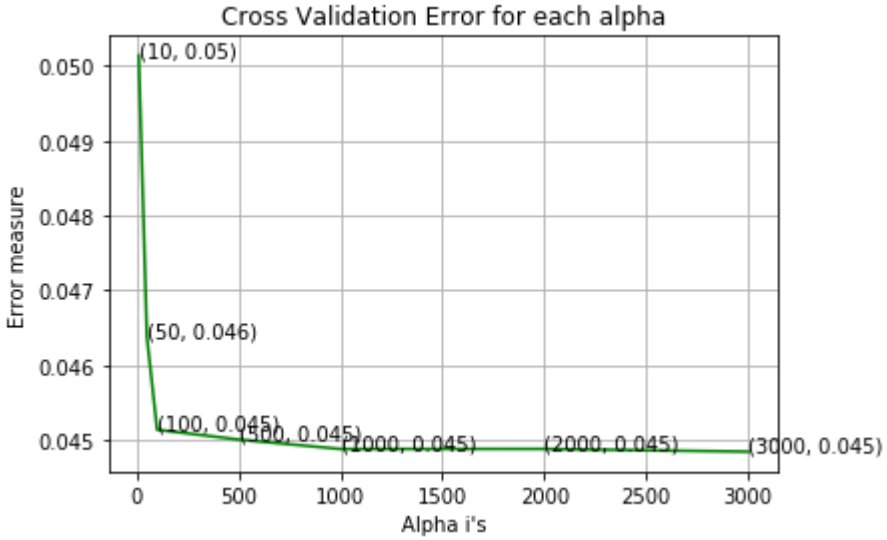

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.4. Random Forest Classifier

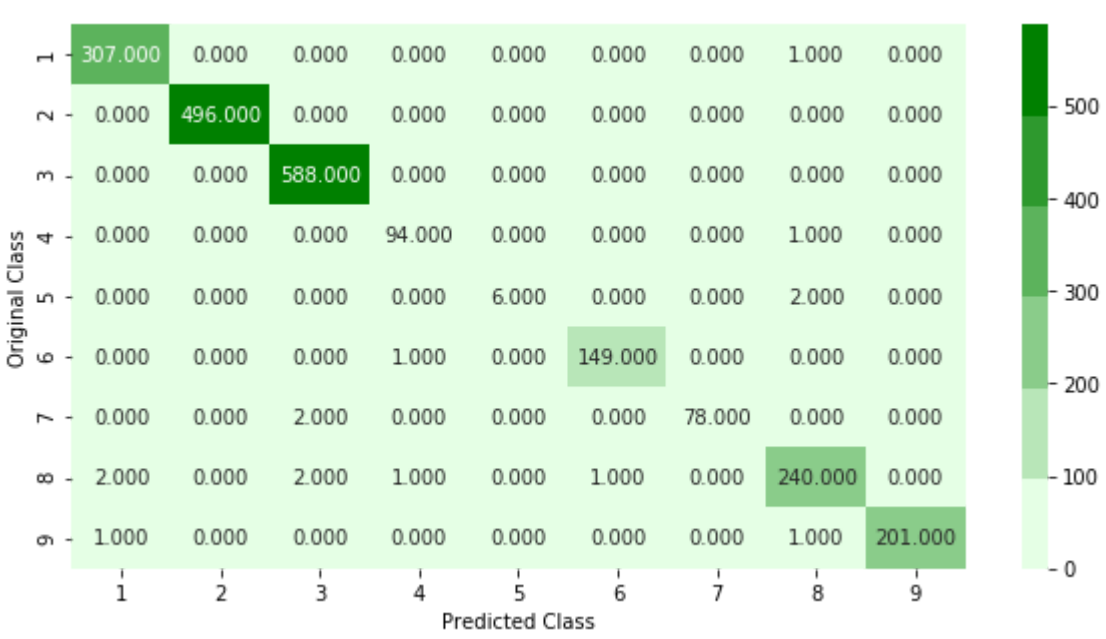
In [0]:
```
%matplotlib inline
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False,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

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))
```

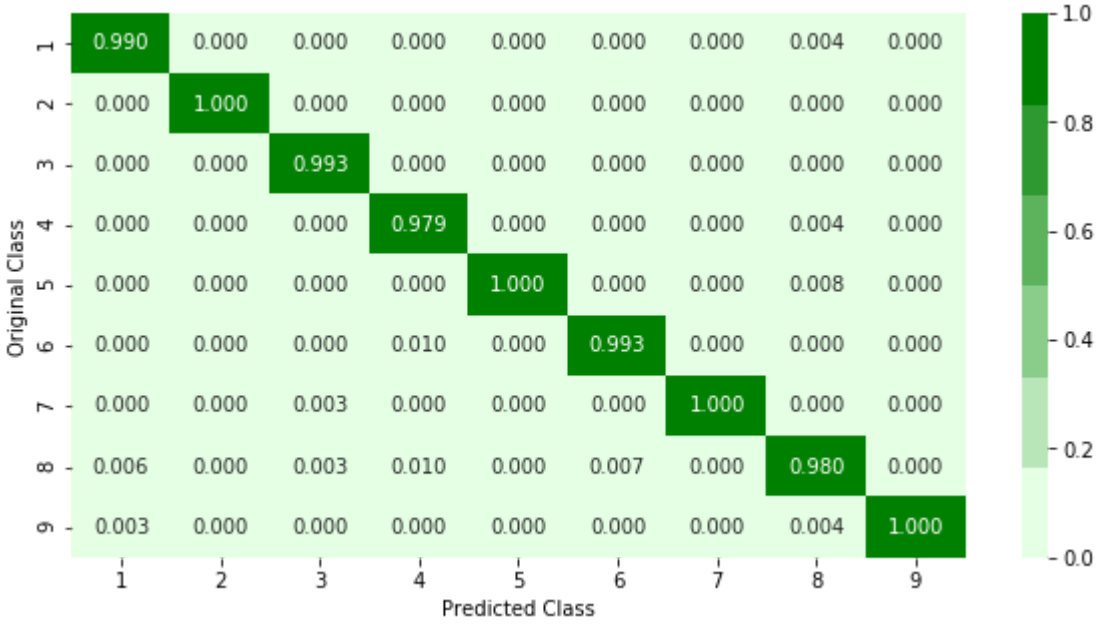
```
log_loss for c =  10 is 0.05013922072841563
log_loss for c =  50 is 0.04638500769388475
log_loss for c =  100 is 0.04513991177672376
log_loss for c =  500 is 0.04500841646529086
log_loss for c =  1000 is 0.044882264063436504
log_loss for c =  2000 is 0.044883179923648465
log_loss for c =  3000 is 0.0448466129883198
```




```
For values of best alpha =  3000 The train log loss is: 0.015580415873834
For values of best alpha =  3000 The cross validation log loss is: 0.0448466129
883198
For values of best alpha =  3000 The test log loss is: 0.04108164323399899
Number of misclassified points  0.6899724011039559
---------------------------------------------------- Confusion matrix -----------
---------------------------------------
```

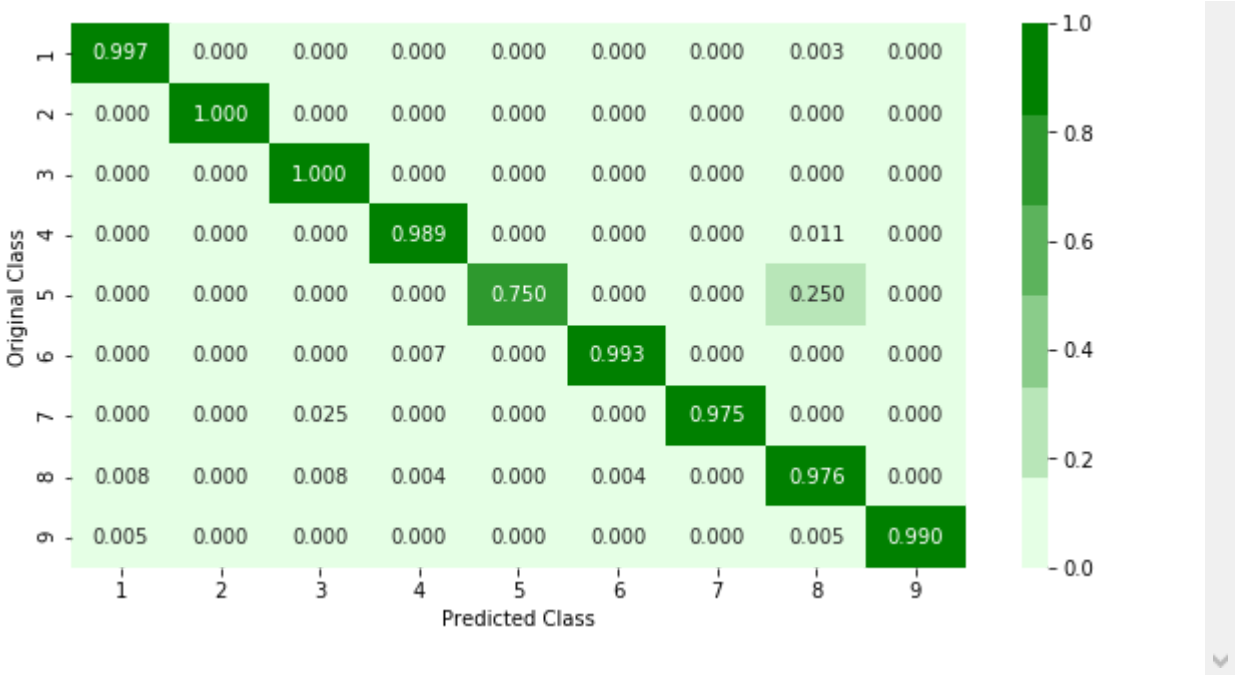


---------- Precision matrix ----------



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

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



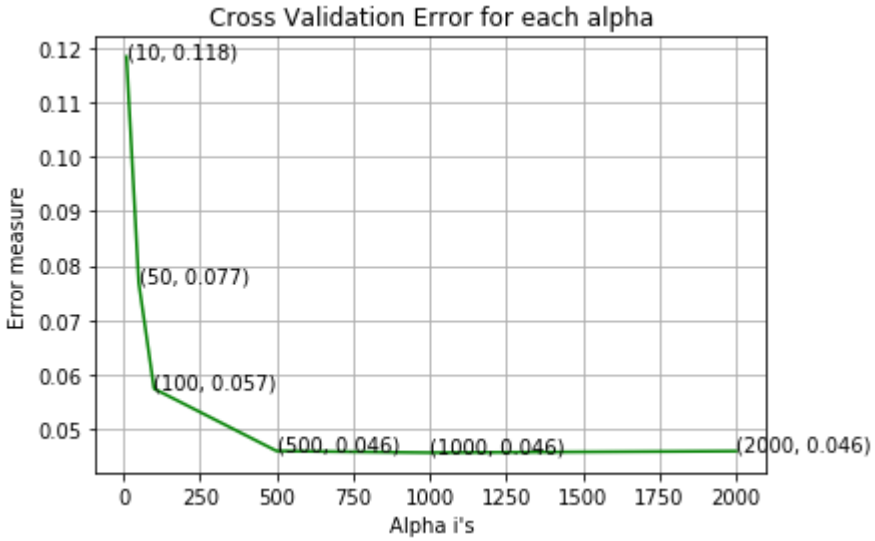

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

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)
```

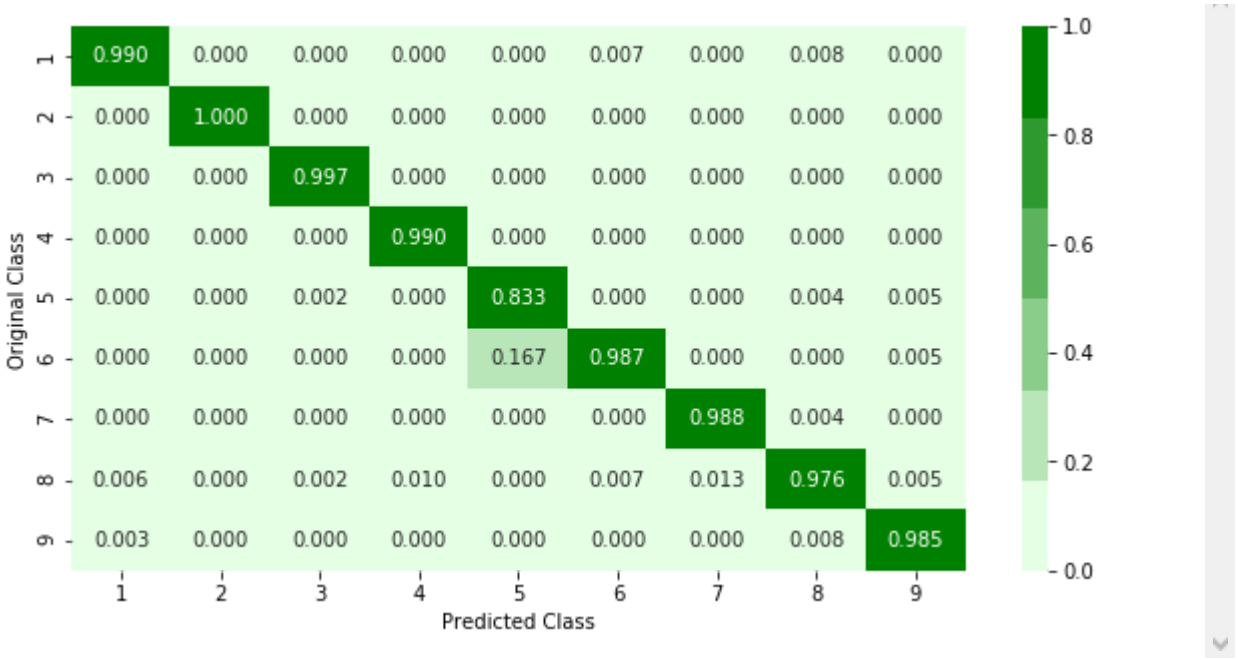
```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.11835135294525914
log_loss for c =  50 is 0.07688720939428521
log_loss for c =  100 is 0.0574208322542204
log_loss for c =  500 is 0.04599429671412565
log_loss for c =  1000 is 0.04568397826437035
log_loss for c =  2000 is 0.0459632196027136
```

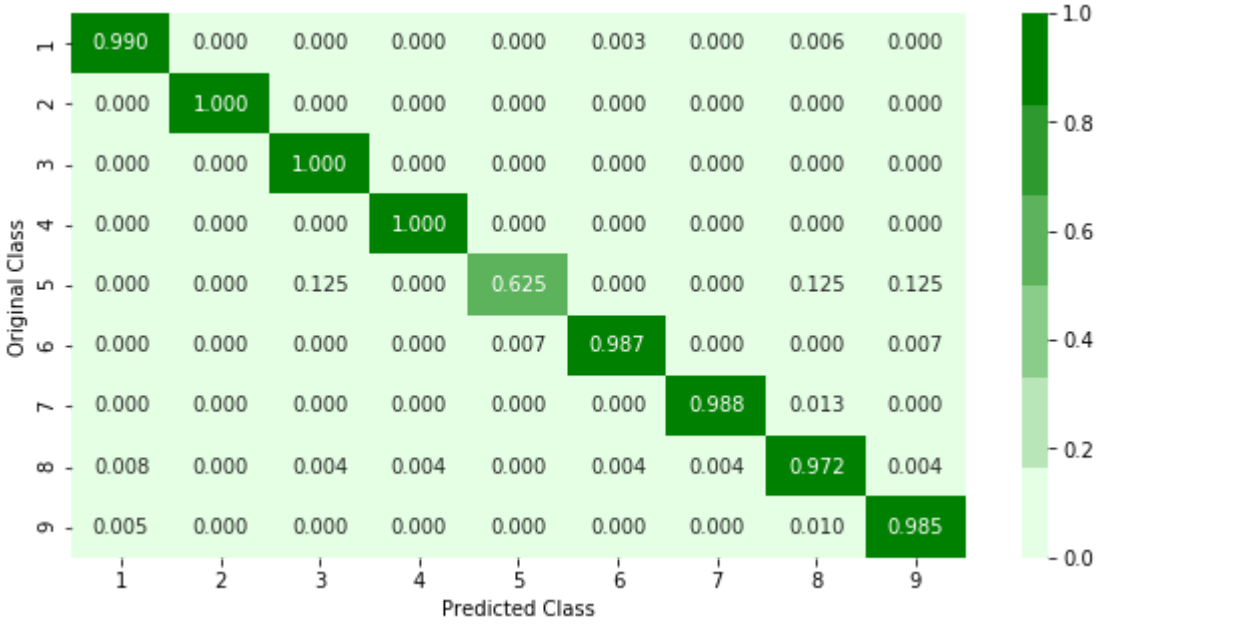



```
For values of best alpha =  1000 The train log loss is: 0.015251753595233744
For values of best alpha =  1000 The cross validation log loss is: 0.0456839782
6437035
For values of best alpha =  1000 The test log loss is: 0.040240729753865515
Number of misclassified points  0.8739650413983441
---------------------------------------------------- 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.
[Parallel(n_jobs=10)]: Done   5 tasks      | elapsed:  4.4min
[Parallel(n_jobs=10)]: Done  15 out of  30 | elapsed:  6.3min remaining:  6.3mi
n
[Parallel(n_jobs=10)]: Done  19 out of  30 | elapsed:  8.3min remaining:  4.8mi
n
[Parallel(n_jobs=10)]: Done  23 out of  30 | elapsed: 10.5min remaining:  3.2mi
n
[Parallel(n_jobs=10)]: Done  27 out of  30 | elapsed: 16.0min remaining:  1.8mi
n
[Parallel(n_jobs=10)]: Done  30 out of  30 | elapsed: 16.2min finished
```

Out[33]:
```
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': 0.5, 'n_estimators': 200, 'max_depth': 10, 'learning_rate': 0.2,
'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=200, learning_rate=0.2, colsample_bytree=0.3, 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))
```

```
train loss 0.015869279639213335
cv loss 0.043125708269246985
test loss 0.03485225395989792
```

In [2]:
```
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", '-', 2.4810, 2.4807])
x.add_row(["KNN", 0.0254, 0.0971, 0.0838])
x.add_row(["LR", 0.2034, 0.2303, 0.1997])
x.add_row(["RF", 0.0155, 0.0155, 0.0410])
x.add_row(["XGBoost", 0.0152, 0.0456, 0.0402])
x.add_row(["XGBoost(Random Search)", 0.0158, 0.0431, 0.0348])
print(x)
```

```
+-----------------------+------------+---------+-----------+
|         Model         | Train Loss | CV Loss | Test Loss |
+-----------------------+------------+---------+-----------+
|      Random Model     |     -      |  2.481  |   2.4807  |
|          KNN          |   0.0254   |  0.0971 |   0.0838  |
|           LR          |   0.2034   |  0.2303 |   0.1997  |
|           RF          |   0.0155   |  0.0155 |   0.041   |
|        XGBoost        |   0.0152   |  0.0456 |   0.0402  |
| XGBoost(Random Search) |   0.0158   |  0.0431 |   0.0348  |
+-----------------------+------------+---------+-----------+
```

Steps Taken

1) Calculated entropy for 256 columns (Opcodes).

2) Image features performed badly so I skipped it.

3) Trained various model using these features.

In [0]: