Loading Libraries

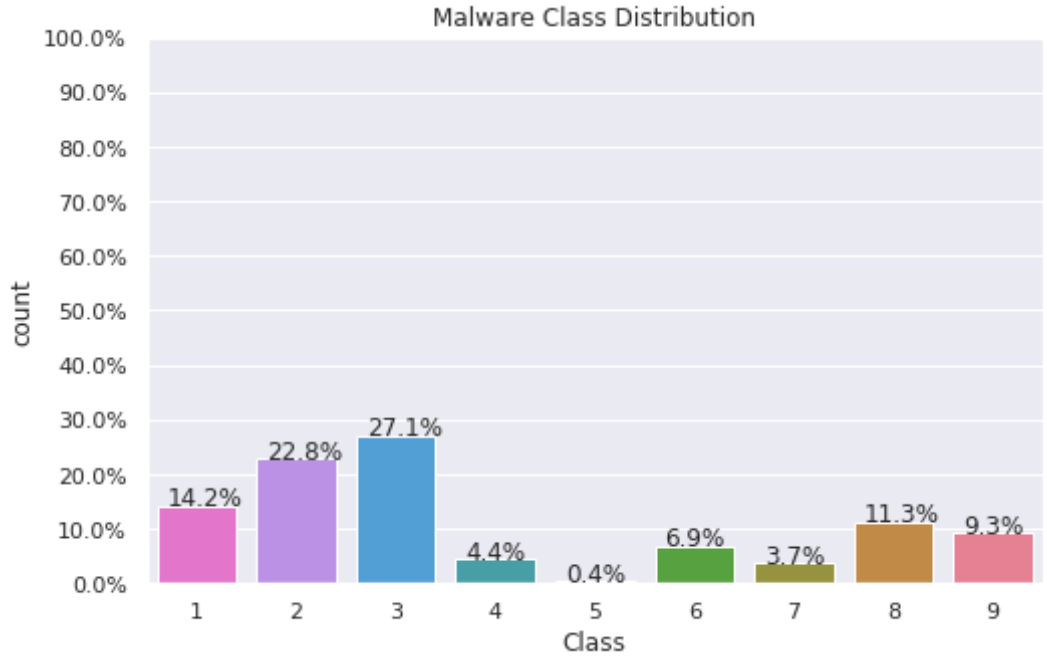
In [0]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import scipy
import pickle
import shutil
import math
import os

import multiprocessing
from multiprocessing import Process

import codecs
import random as r

from xgboost import XGBClassifier
from sklearn.manifold import TSNE
from sklearn import preprocessing
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
```

```
matplotlib.use(u'nbAgg')
```

In [0]:
```
# Separating asm and byte file

if not os.path.isdir('../byteFiles'):
    os.makedirs('../byteFiles')

if os.path.isdir('../train'):
    os.rename('../train', '../asmFiles')
    data_files = os.listdir('../asmFiles')
    for file in data_files:
        if file.endswith("bytes"):
            shutil.move('../asmFiles/' + file, '../byteFiles')
```

In [0]:
```
labels = pd.read_csv('./trainLabels.csv')
```

In [0]:
```
sns.set()
plt.figure(figsize = (8, 5))
ax = sns.countplot(x = 'Class', data = labels, palette = reversed(sns.c
olor_palette('husl', 9)))
for p in ax.patches:
    ax.annotate('{:.1f}%'.format(100*p.get_height() / labels.shape[0]),
 (p.get_x() + 0.1, p.get_height() + 5))

ax.yaxis.set_ticks(np.linspace(0, labels.shape[0], 11))
ax.set_yticklabels(map('{:.1f}%'.format, 100 * ax.yaxis.get_majorticklo
cs() / labels.shape[0]))
ax.set_title('Malware Class Distribution')
plt.show()
```



File size of byte files as a feature

In [0]:
```
byte_files = os.listdir('../byteFiles')
file_names = labels['Id'].tolist()
class_labels = labels['Class'].tolist()
class_bytes = []
size_bytes = []
fnames = []

for file in byte_files:
    statinfo = os.stat('../byteFiles/' + file)
    file = file.split('.')[0]

    if any(file == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_labels[i])
```

```
        size_bytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

data_size_byte = pd.DataFrame({'ID':fnames,'size':size_bytes,'Class':cl
ass_bytes})
data_size_byte.head()
```

Out[0]:

	Class	ID	size
0	2	j4LbkH0Dx8eXOJnPBtZT	0.000000
1	2	1IqjvX7dTxCu54M0ylBE	2.753906
2	1	CoNnhm2Se85BAHOs1kJD	1.300781
3	9	c0Hj14EGaFBbPnefqS78	0.644531
4	9	a4mHXyRFw3jUAgcdq6Op	0.644531

feature extraction from byte files(bag of words)

In [0]:

```
#contents of .byte files
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#--------------------
#remove the starting address 00401000
byte_files = os.listdir('../byteFiles')
file_names = []
array = []

for file in byte_files:
    if(file.endswith("bytes")):
        file = file.split('.')[0]
        text_file = open('../byteFiles/' + file + ".txt", 'w+')
        with open('../byteFiles/' + file + ".bytes", "r") as fp:
            lines = ""
            for line in fp:
                a = line.rstrip().split(" ")[1:]
                b = ' '.join(a)
                b = b + "\n"
                text_file.write(b)
```

```
            fp.close()
            os.remove('../byteFiles/' + file + '.bytes')
        text_file.close()

byte_files = os.listdir('../byteFiles')
file_names2 = []
feature_matrix = np.zeros((len(byte_files), 257), dtype = int)
k = 0

#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file = open('result.csv', 'w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,1
1,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,
29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,4
0,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,
58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6
f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,
87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9
e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,
b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,c
d,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,
e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,f
c,fd,fe,ff,??")

for file in byte_files:
    file_names2.append(file)
    byte_feature_file.write(file + ",")
    if(file.endswith("txt")):
        with open('../byteFiles/' + file, "r") as byte_file:
            for lines in byte_file:
                line = lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code == '??':
                        feature_matrix[k][256] += 1
                    else:
                        feature_matrix[k][int(hex_code, 16)] += 1
        byte_file.close()
```

```
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i) + ",")
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [0]:
```
byte_features = pd.read_csv("../result.csv")
byte_features.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	f7
0	1IqjvX7dTxCu54M0ylBE	264506	4509	2981	7938	3619	2868	4168	3174	3070	...	325
1	CoNnhm2Se85BAHOs1kJD	58692	2140	842	965	1333	623	621	795	3552	...	6043
2	c0Hj14EGaFBbPnefqS78	93102	742	525	571	526	582	423	434	587	...	456
3	a4mHXyRFw3jUAgcdq6Op	90719	691	434	545	454	731	440	450	593	...	426
4	BcuE9gIzJoh70dAlbVqw	11449	5561	3193	3436	3156	3268	3220	3301	3184	...	3257

5 rows × 258 columns

In [0]:
```
result = pd.merge(byte_features, data_size_byte, on = 'ID', how = 'lef
t')
result.head()
```

Out[0]:

	ID	0	1	2	3	4	5	6	7	8	...	f9
0	1IqjvX7dTxCu54M0ylBE	264506	4509	2981	7938	3619	2868	4168	3174	3070	...	242
1	CoNnhm2Se85BAHOs1kJD	58692	2140	842	965	1333	623	621	795	3552	...	4779
2	c0Hj14EGaFBbPnefqS78	93102	742	525	571	526	582	423	434	587	...	436
3	a4mHXyRFw3jUAgcdq6Op	90719	691	434	545	454	731	440	450	593	...	398
4	BcuE9gIzJoh70dAlbVqw	11449	5561	3193	3436	3156	3268	3220	3301	3184	...	3153

5 rows × 260 columns

In [0]:
```
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != 'ID' and str(feature_name) != 'Class'
):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (m
ax_value - min_value)
    return result1
result = normalize(result)
```

In [0]:
```
data_y = result['Class']
result.head()
```

Out[0]:

	ID	0	1	2	3	4	5	
0	1IqjvX7dTxCu54M0ylBE	0.115489	0.006348	0.001659	0.004282	0.002215	0.001623	0.00235
1	CoNnhm2Se85BAHOs1kJD	0.025626	0.003013	0.000468	0.000521	0.000816	0.000353	0.00035
2	c0Hj14EGaFBbPnefqS78	0.040650	0.001045	0.000292	0.000308	0.000322	0.000329	0.00023
3	a4mHXyRFw3jUAgcdq6Op	0.039610	0.000973	0.000241	0.000294	0.000278	0.000414	0.00024
4	BcuE9gIzJoh70dAlbVqw	0.004999	0.007829	0.001776	0.001854	0.001932	0.001850	0.00181

5 rows × 260 columns

Training-set and Test-set splite

In [0]:
```
x_train, x_test, y_train, y_test = train_test_split(result.drop(['ID',
'Class'], axis=1), data_y, stratify = data_y, test_size = 0.20)
```

```
x_trn, x_cv, y_trn, y_cv = train_test_split(x_train, y_train, stratify
= y_train, test_size = 0.20)
```

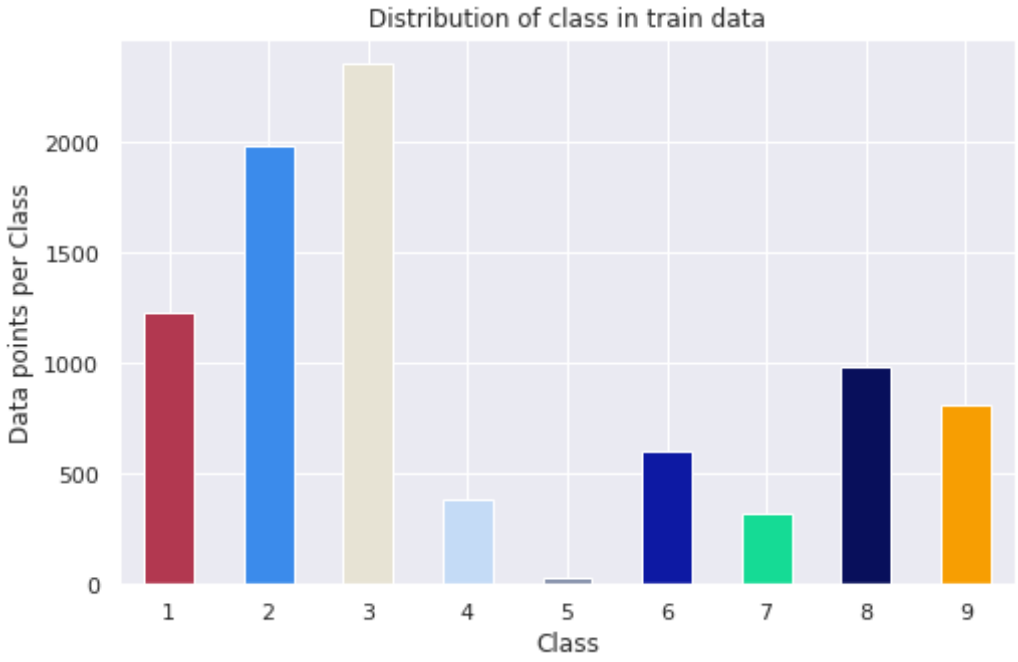
In [0]:
```
print('Number of data points in train data:', x_train.shape[0])
print('Number of data points in test data:', x_test.shape[0])
print('Number of data points in cross validation data:', x_cv.shape[0])
```

```
Number of data points in train data: 8693
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

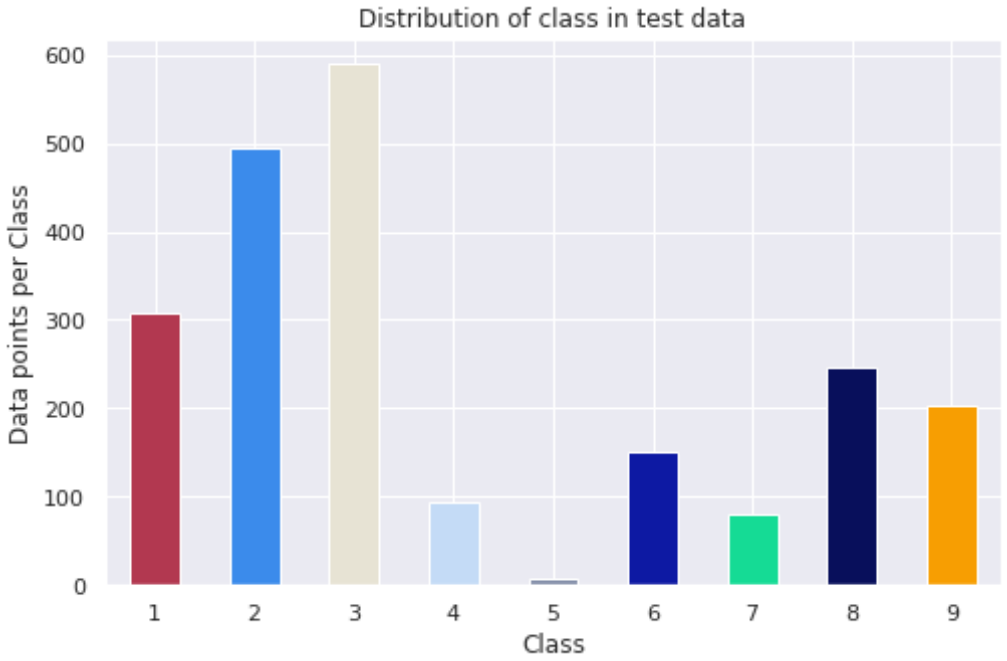
Train and test Distribution

In [0]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0
d19a3', '#15db95', '#080f5b', '#f79e02']
ax = train_class_distribution.plot(kind = 'bar', color = my_colors, rot
 = 0)
ax.figure.set_size_inches(8, 5)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of class in train data')
plt.grid(b = True)
plt.show()

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i + 1, ':', train_class_dis
tribution.values[i], '(', np.round((train_class_distribution.values[i]
/ y_train.shape[0]*100), 3), '%)')

print('-'*80)
ax = test_class_distribution.plot(kind = 'bar', color = my_colors, rot
```

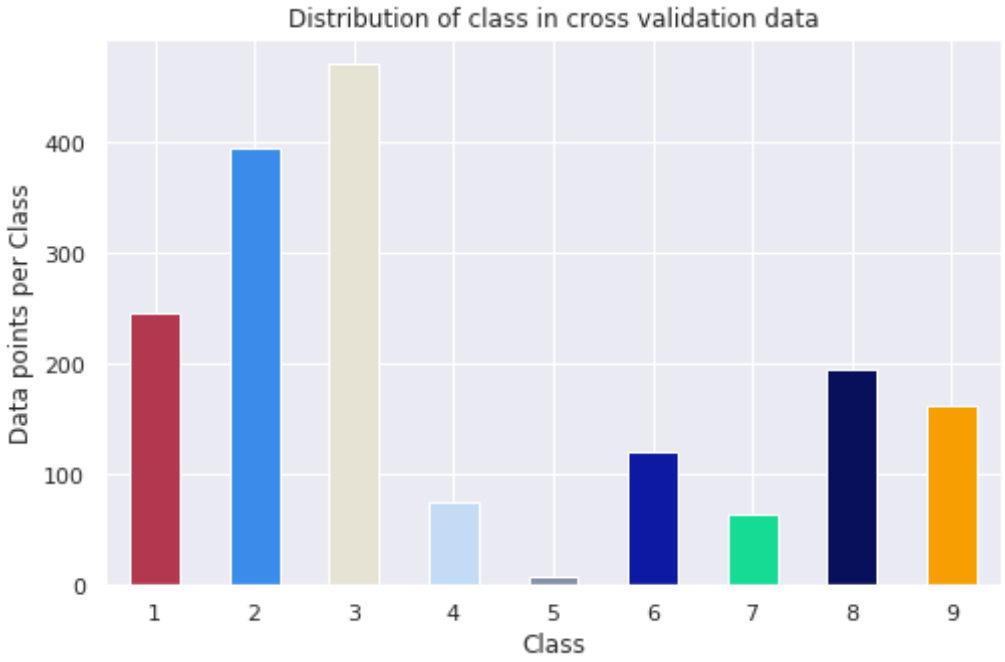
```
= 0)
ax.figure.set_size_inches(8, 5)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of class in test data')
plt.grid(b = True)
plt.show()

sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i + 1, ':', test_class_dist
ribution.values[i], '(', np.round((test_class_distribution.values[i] /
y_test.shape[0]*100), 3), '%)')

print('-'*80)
ax = cv_class_distribution.plot(kind = 'bar', color = my_colors, rot =
0)
ax.figure.set_size_inches(8, 5)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of class in cross validation data')
plt.grid(b = True)
plt.show()

sorted_yi = np.argsort(-cv_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i + 1, ':', cv_class_distri
bution.values[i], '(', np.round((cv_class_distribution.values[i] / y_cv
.shape[0]*100), 3), '%)')
```




```
Number of data points in class 3 : 2353 ( 27.068 %)
Number of data points in class 2 : 1982 ( 22.8 %)
Number of data points in class 1 : 1233 ( 14.184 %)
Number of data points in class 8 : 982 ( 11.296 %)
Number of data points in class 9 : 810 ( 9.318 %)
Number of data points in class 6 : 601 ( 6.914 %)
Number of data points in class 4 : 380 ( 4.371 %)
Number of data points in class 7 : 318 ( 3.658 %)
Number of data points in class 5 : 34 ( 0.391 %)
------------------------------------------------------------------------
-----------
```




```
Number of data points in class 3 : 589 ( 27.093 %)
Number of data points in class 2 : 495 ( 22.769 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
------------------------------------------------------------------------
-----------
```




```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

plotting functions

In [0]:
```
def err_metrics(y, yhat):

    confuzn_mtx = confusion_matrix(y, yhat)
    precision_mtx = confuzn_mtx / confuzn_mtx.sum(axis = 0)
    recall_mtx = (confuzn_mtx.T / confuzn_mtx.sum(axis = 1)).T
    labels = np.unique(y)

    print('-'*40, 'CONFUSION-MATRIX', '-'*40)
    plt.figure(figsize=(18, 7))
    sns.heatmap(confuzn_mtx, annot=True, cmap=plt.cm.get_cmap('Blues',
9), fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print('-'*40, 'PRECISION-MATRIX', '-'*40)
    plt.figure(figsize=(18, 7))
    sns.heatmap(precision_mtx, annot=True, cmap=plt.cm.get_cmap('Blues'
, 9), fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print('-'*40, 'RECALL-MATRIX', '-'*40)
    plt.figure(figsize=(18, 7))
    sns.heatmap(recall_mtx, annot=True, cmap=plt.cm.get_cmap('Blues', 9
), fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [0]:
```
def err_compare(train_loss, cv_loss, alpha, hyp):

    sns.set()
    plt.figure(1)
```

```
    plt.figure(figsize = (8, 5))
    plt.plot(alpha, cv_loss, label = 'cv_error', color = 'Red')
    plt.plot(alpha, train_loss, label = 'train_error', color = 'Blue')
    if hyp == 'Alpha':
        plt.xscale('log')
    plt.xlabel(hyp + '-Values')
    plt.ylabel('Error Values')
    plt.legend()
    plt.title('CV & TRAIN-ERR')
    plt.show()
```

ML Modeling On Byte Features

- **Random Model**

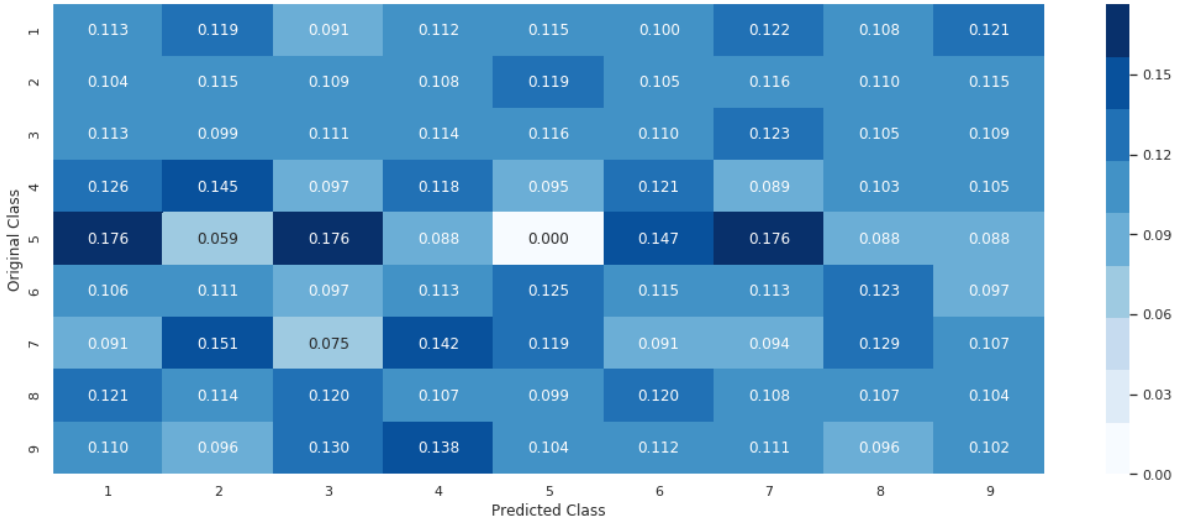
In [0]:
```
def random_clf(ytrain, ytest):

    ytrain_dummy = np.zeros((1, len(np.unique(ytrain))))
    ytest_dummy = np.zeros((1, len(np.unique(ytest))))

    for i in range(len(ytrain)):
        random_probs = np.random.rand(1, len(np.unique(ytrain)))
        ytrain_dummy = np.vstack([ytrain_dummy, random_probs / random_p
robs.sum()])
    print('Log-Loss on Train-set using Random Model is ', log_loss(ytra
in, ytrain_dummy[1:, :]))
    err_metrics(ytrain, np.argmax(ytrain_dummy[1:, :], axis = 1) + 1)

    for i in range(len(ytest)):
        random_probs = np.random.rand(1, len(np.unique(ytest)))
        ytest_dummy = np.vstack([ytest_dummy, random_probs / random_pro
bs.sum()])
    print('Log-Loss on Test-set using Random Model is ', log_loss(ytest
, ytest_dummy[1:, :]))
    err_metrics(ytest, (np.argmax(ytest_dummy[1:, :], axis = 1)) + 1)
```

In [0]:
```
random_clf(y_train, y_test)
```

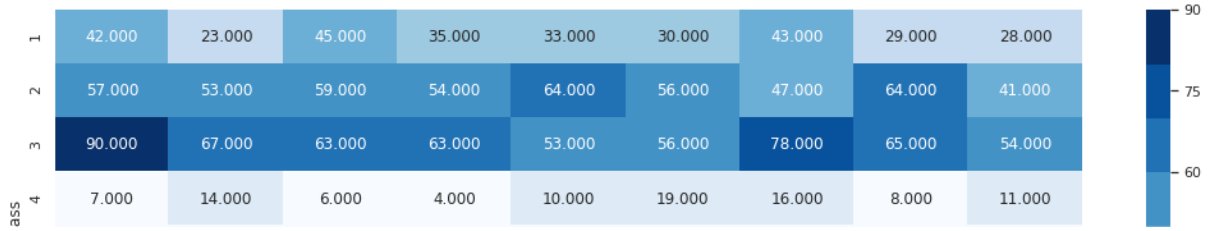
```
Log-Loss on Train-set using Random Model is  2.4851206045937735
---------------------------------------- CONFUSION-MATRIX ----------
------------------------------
```

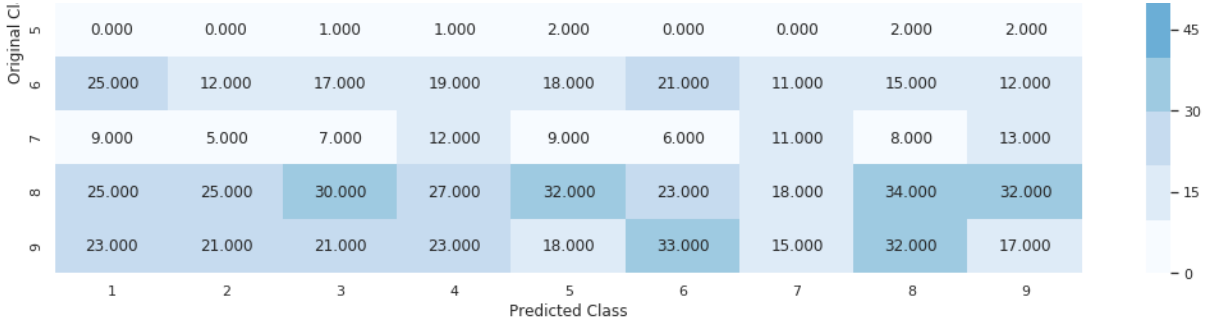

```
---------------------------------------- PRECISION-MATRIX ----------
------------------------------
```

```
---------------------------------------- RECALL-MATRIX --------------
---------------------------
```

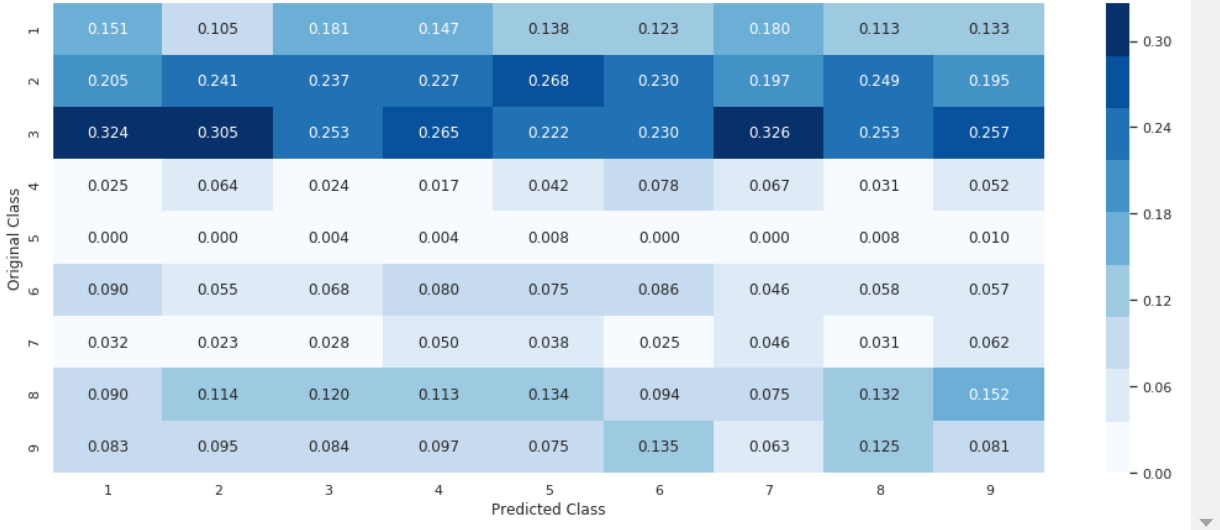



```
Log-Loss on Test-set using Random Model is  2.4726170397689335
---------------------------------------- CONFUSION-MATRIX -----------
-----------------------------
```

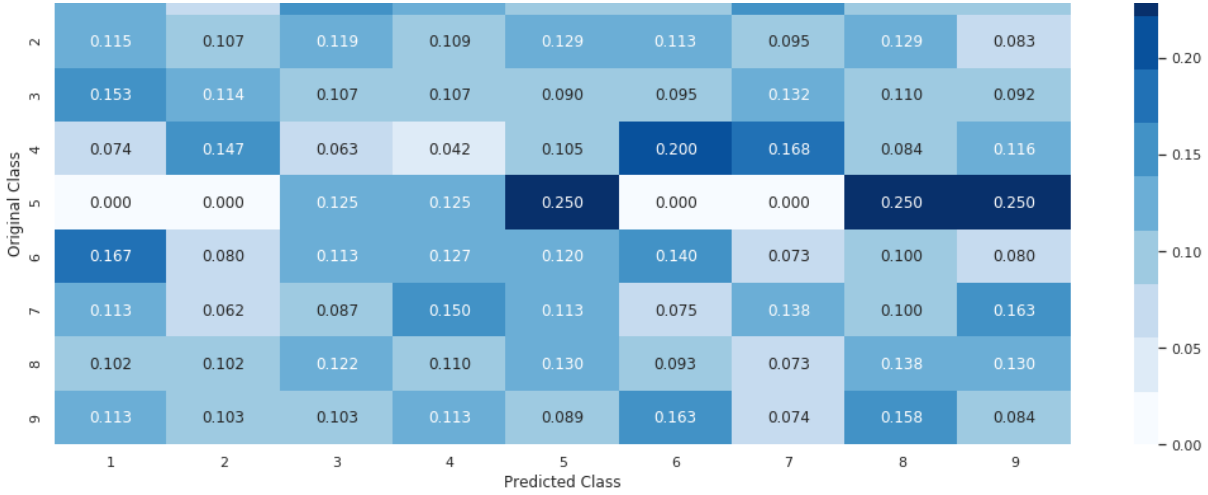




---------- PRECISION-MATRIX ----------



---------- RECALL-MATRIX ----------



- **KNN Model**

In [0]:
```
def knn_model(operation, best_k = None):

    k_val = np.arange(1, 15, 2)
    if operation == 'Training':
        cv_err = []
        train_err = []
        for k in k_val:
            clf = KNeighborsClassifier(n_neighbors = k, n_jobs = -1)
            clf.fit(x_trn, y_trn)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn, y_trn)
            pred_proba_trn = sig_clf.predict_proba(x_trn)
            pred_proba_cv = sig_clf.predict_proba(x_cv)
            train_err.append(log_loss(y_trn, pred_proba_trn, labels = c
lf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv, pred_proba_cv, labels = clf.cl
asses_, eps = 1e-15))
```

```
        err_compare(train_err, cv_err, k_val, 'K')
    else:
        clf = KNeighborsClassifier(n_neighbors = best_k, n_jobs = -1)
        clf.fit(x_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train, y_train)
        pred_proba_trn = sig_clf.predict_proba(x_train)
        pred_proba_tst = sig_clf.predict_proba(x_test)
        trn_err = clf.score(x_train, y_train)
        tst_err = clf.score(x_test, y_test)
        print('Log-Loss for Train-set is :', log_loss(y_train, pred_pro
ba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test, pred_proba
_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test, sig_clf.predict(x_test))
```

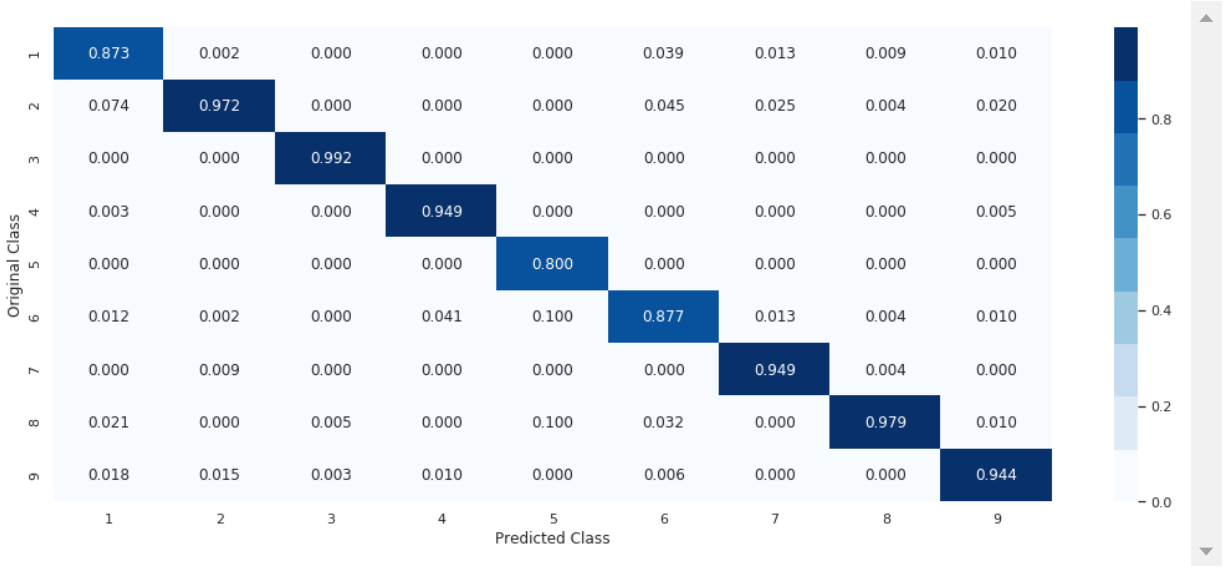
In [0]:
```
knn_model('Training')
```

```
<matplotlib.figure.Figure at 0x7f890edccdd8>
```

In [0]:
```
knn_model('Testing', 3) # Performance Metric
```

```
Log-Loss for Train-set is : 0.11825387663782871
Log-Loss for Test-set is : 0.20408219520299187
Train Accuracy is : 0.9690555619463936
Test Accuracy is : 0.9549218031278749
---------------------------------------- CONFUSION-MATRIX ----------
----------------------------
```

---------- PRECISION-MATRIX ----------



---------- RECALL-MATRIX ----------

- **Logistic Regression Model**

In [0]:
```
def lr_model(operation, best_alpha = None):

    alpha = [10 ** x for x in range(-4, 4)]
    if operation == 'Training':
        cv_err = []
        train_err = []
        for a in alpha:
            clf = LogisticRegression(penalty = 'l2', C = a, class_weigh
t = 'balanced', n_jobs = -1)
            clf.fit(x_trn, y_trn)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn, y_trn)
            pred_proba_trn = sig_clf.predict_proba(x_trn)
            pred_proba_cv = sig_clf.predict_proba(x_cv)
            train_err.append(log_loss(y_trn, pred_proba_trn, labels = c
lf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv, pred_proba_cv, labels = clf.cl
asses_, eps = 1e-15))
        err_compare(train_err, cv_err, alpha, 'Alpha')
```

```
    else:
        clf = LogisticRegression(penalty = 'l2', C = best_alpha, class_
weight = 'balanced', n_jobs = -1)
        clf.fit(x_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train, y_train)
        pred_proba_trn = sig_clf.predict_proba(x_train)
        pred_proba_tst = sig_clf.predict_proba(x_test)
        trn_err = clf.score(x_train, y_train)
        tst_err = clf.score(x_test, y_test)
        print('Log-Loss for Train-set is :', log_loss(y_train, pred_pro
ba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test, pred_proba
_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test, sig_clf.predict(x_test))
```

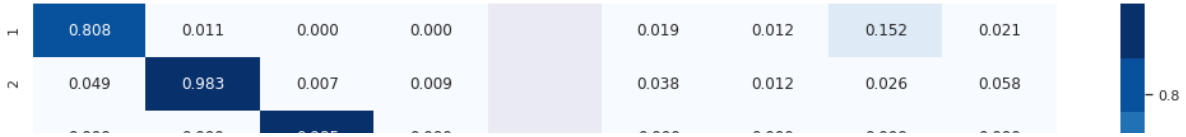
In [0]:
```
lr_model('Training')
```

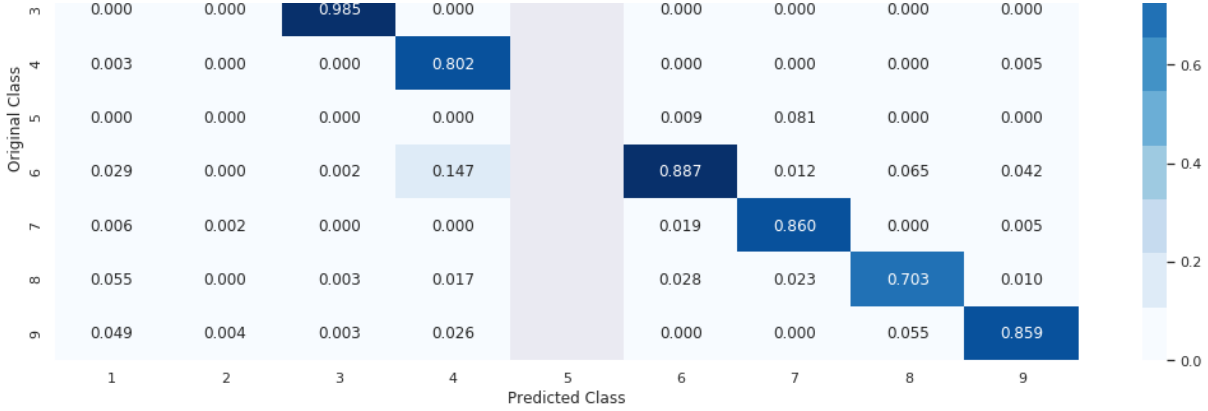
```
<matplotlib.figure.Figure at 0x7f890eb67198>
```

In [0]:
```
lr_model('Testing', 100) #Performance Metric
```

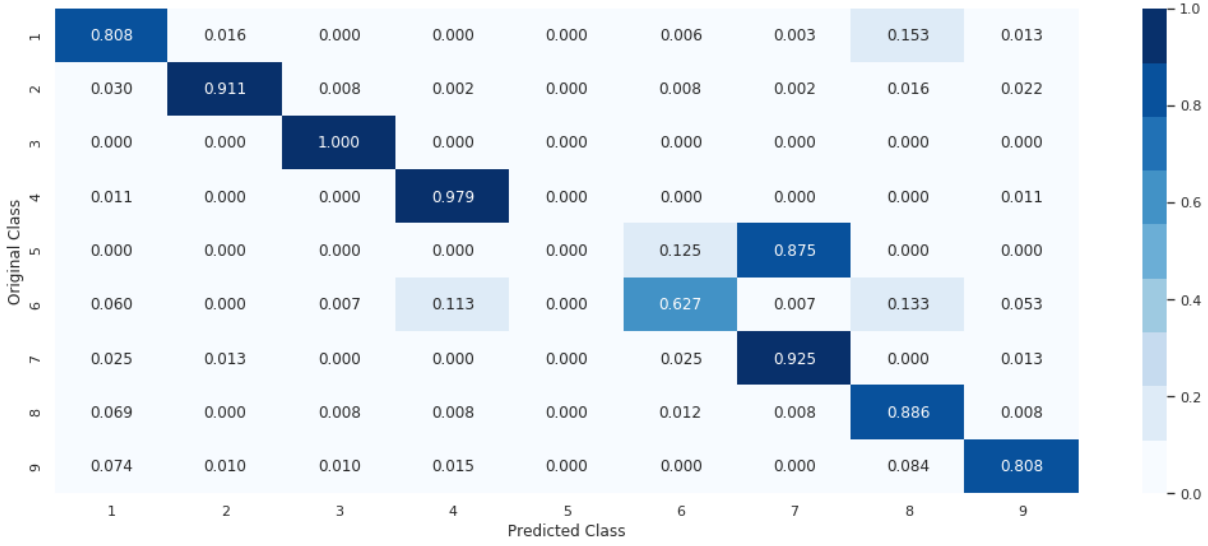
```
Log-Loss for Train-set is : 0.4928819297247423
Log-Loss for Test-set is : 0.53093612806751
Train Accuracy is : 0.898078914068791
Test Accuracy is : 0.8877644894204232
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------



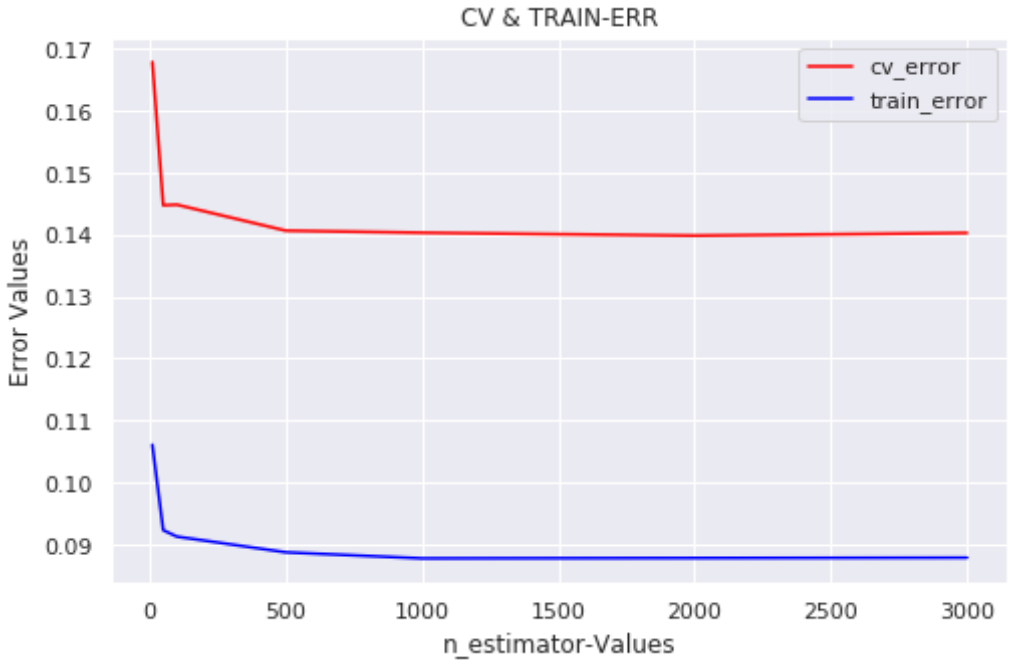
- **Decision Tree Model**

In [0]:
```
def dt_model(operation, best_n = None):

    n_base = [10,50,100,500,1000,2000,3000]
```

```
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = RandomForestClassifier(n_estimators = n, max_depth =
8, random_state = 12, n_jobs = -1)
            clf.fit(x_trn, y_trn)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn, y_trn)
            pred_proba_trn = sig_clf.predict_proba(x_trn)
            pred_proba_cv = sig_clf.predict_proba(x_cv)
            train_err.append(log_loss(y_trn, pred_proba_trn, labels = c
lf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv, pred_proba_cv, labels = clf.cl
asses_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = RandomForestClassifier(n_estimators = best_n, max_depth =
 8, random_state = 12, n_jobs = -1)
        clf.fit(x_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train, y_train)
        pred_proba_trn = sig_clf.predict_proba(x_train)
        pred_proba_tst = sig_clf.predict_proba(x_test)
        trn_err = clf.score(x_train, y_train)
        tst_err = clf.score(x_test, y_test)
        print('Log-Loss for Train-set is :', log_loss(y_train, pred_pro
ba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test, pred_proba
_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test, sig_clf.predict(x_test))
```

In [0]:
```
dt_model('Training')
```

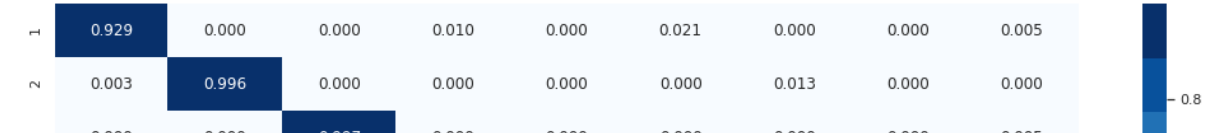
```
<matplotlib.figure.Figure at 0x7f890ef4a710>
```

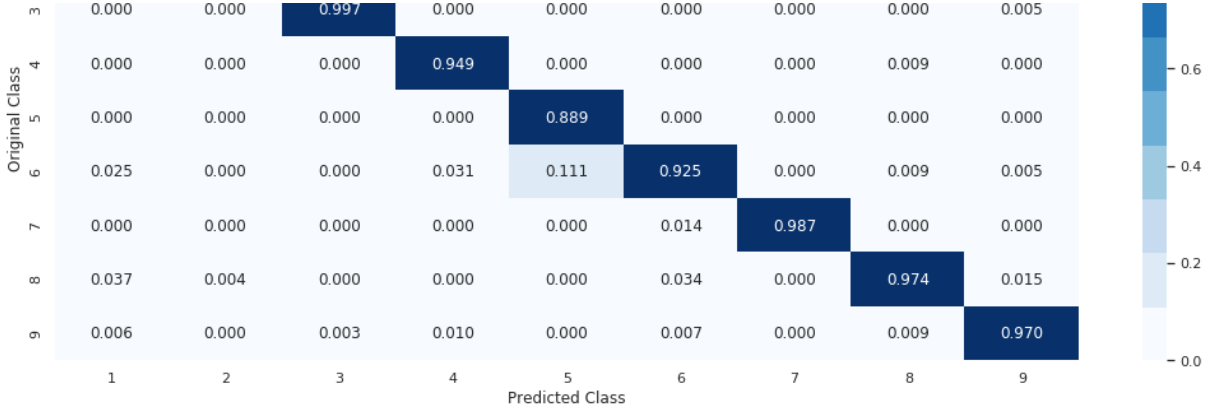


In [0]:
```
dt_model('Testing', 1000) #Performance Metric
```

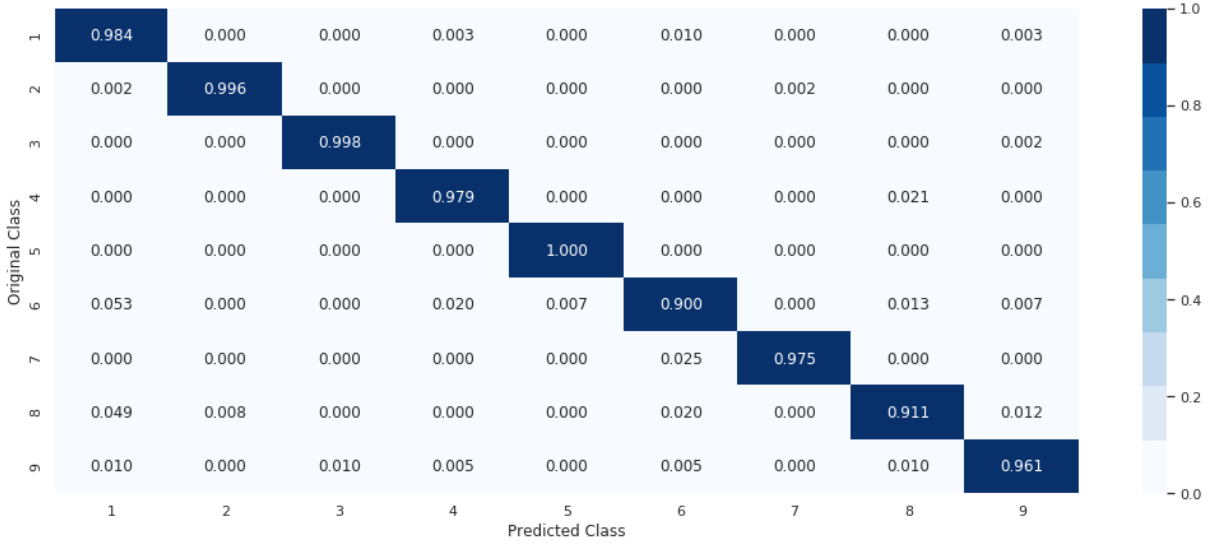
```
Log-Loss for Train-set is : 0.08996798105986126
Log-Loss for Test-set is : 0.12080088512824226
Train Accuracy is : 0.9731968250316346
Test Accuracy is : 0.9659613615455381
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------



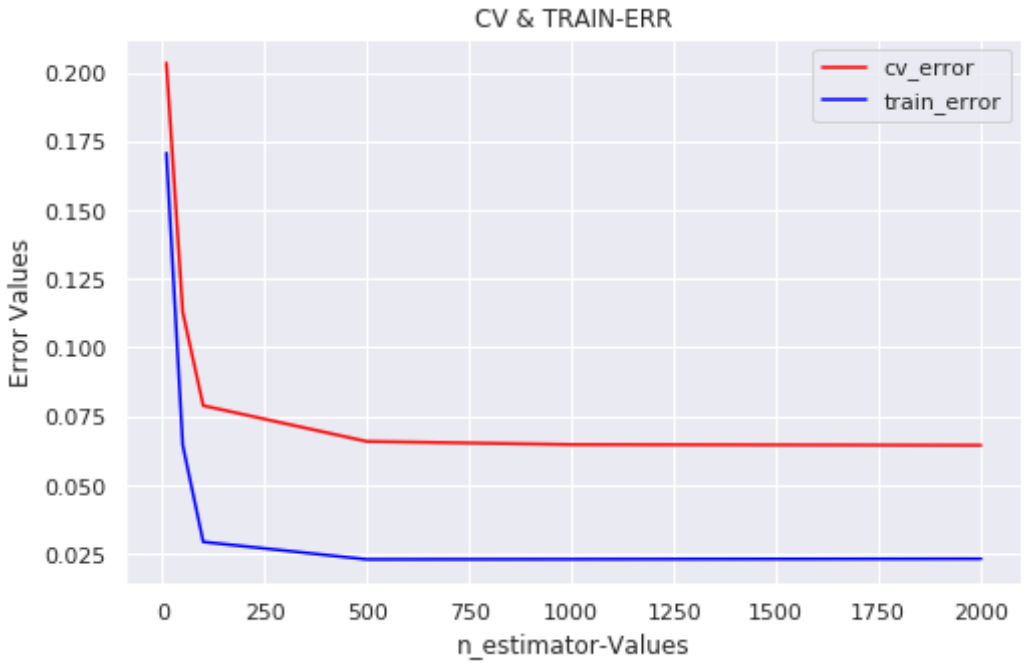
- **XGBOOST Model**

In [0]:
```
def xgb_model(operation, best_n = None):

    n_base = [10, 50, 100, 500, 1000, 2000]
```

```
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = XGBClassifier(n_estimators = n, nthread = -1)
            clf.fit(x_trn, y_trn)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn, y_trn)
            pred_proba_trn = sig_clf.predict_proba(x_trn)
            pred_proba_cv = sig_clf.predict_proba(x_cv)
            train_err.append(log_loss(y_trn, pred_proba_trn, labels = c
lf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv, pred_proba_cv, labels = clf.cl
asses_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = XGBClassifier(n_estimators = best_n, nthread = -1)
        clf.fit(x_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train, y_train)
        pred_proba_trn = sig_clf.predict_proba(x_train)
        pred_proba_tst = sig_clf.predict_proba(x_test)
        trn_err = clf.score(x_train, y_train)
        tst_err = clf.score(x_test, y_test)
        print('Log-Loss for Train-set is :', log_loss(y_train, pred_pro
ba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test, pred_proba
_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test, sig_clf.predict(x_test))
```

In [0]:
```
xgb_model('Training')
```

```
<matplotlib.figure.Figure at 0x7f892ba89668>
```

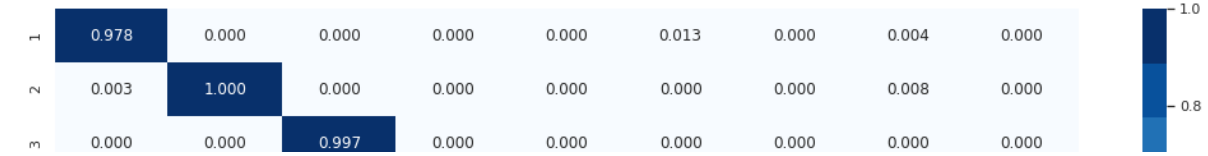


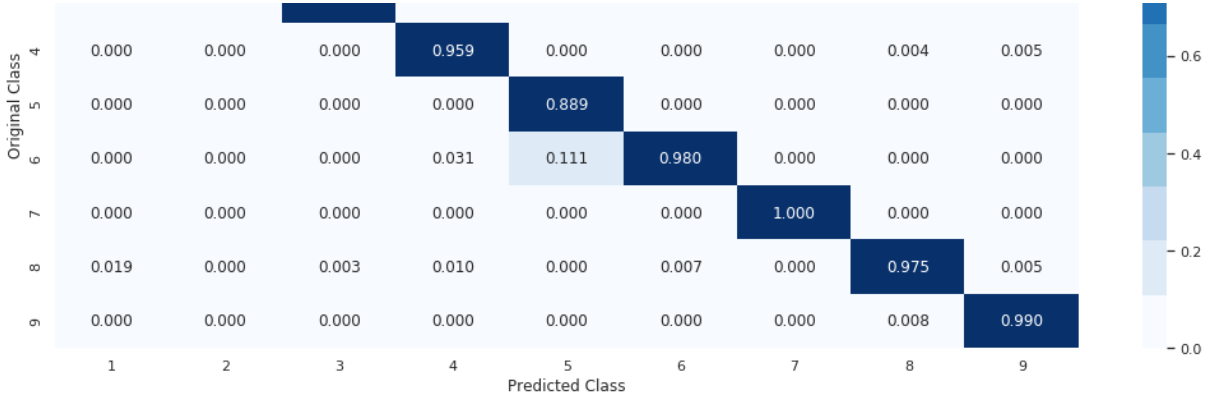
In [0]:
```
xgb_model('Testing', 500) #Performance Metric
```

```
/home/sradheya/anaconda3/lib/python3.6/site-packages/sklearn/preprocess
ing/label.py:151: DeprecationWarning: The truth value of an empty array
is ambiguous. Returning False, but in future this will result in an err
or. Use `array.size > 0` to check that an array is not empty.
  if diff:
/home/sradheya/anaconda3/lib/python3.6/site-packages/sklearn/preprocess
ing/label.py:151: DeprecationWarning: The truth value of an empty array
is ambiguous. Returning False, but in future this will result in an err
or. Use `array.size > 0` to check that an array is not empty.
  if diff:
```

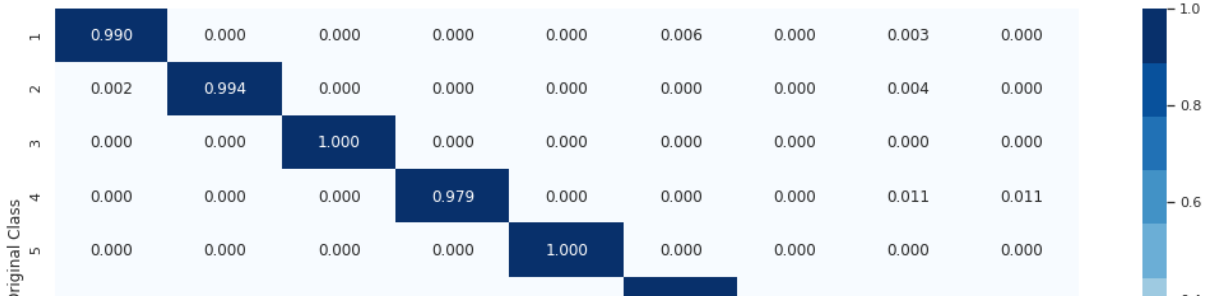
```
Log-Loss for Train-set is : 0.019872456387703585
Log-Loss for Test-set is : 0.061377559778410064
Train Accuracy is : 1.0
Test Accuracy is : 0.9894204231830727
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

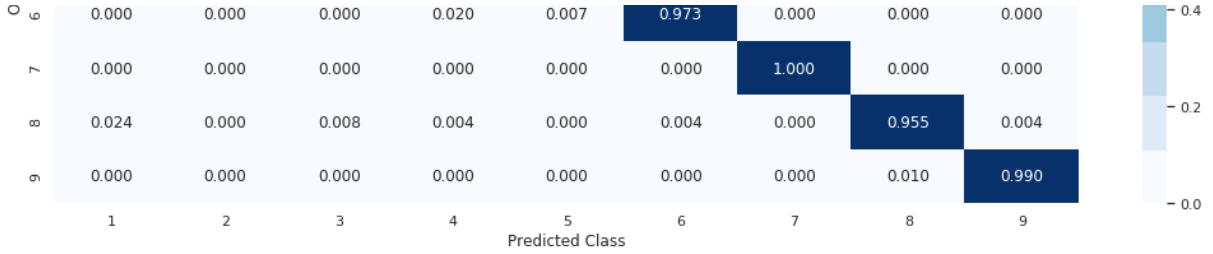
---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------





- **hyperparameter tuning XGBOOST**

In [0]:
```
import sys
import warnings

if not sys.warnoptions:
    warnings.simplefilter("ignore")

def xgb_tune():
    clf = XGBClassifier()

    prams={
        'learning_rate':[0.01, 0.03, 0.05, 0.1, 0.15, 0.2],
        'n_estimators':[100, 200, 500, 1000],
        'max_depth':[3, 5, 8],
        'colsample_bytree':[0.1, 0.3, 0.5, 1],
        'subsample':[0.1, 0.3, 0.5, 1]
    }
    random_clf = RandomizedSearchCV(clf, param_distributions = prams, v
erbose = 1, n_jobs = -1)
    random_clf.fit(x_train, y_train)
    print(random_clf.best_params_)
```

In [0]:
```
xgb_tune()
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  8.3min finished
{'subsample': 0.3, 'n_estimators': 500, 'max_depth': 3, 'learning_rat
```

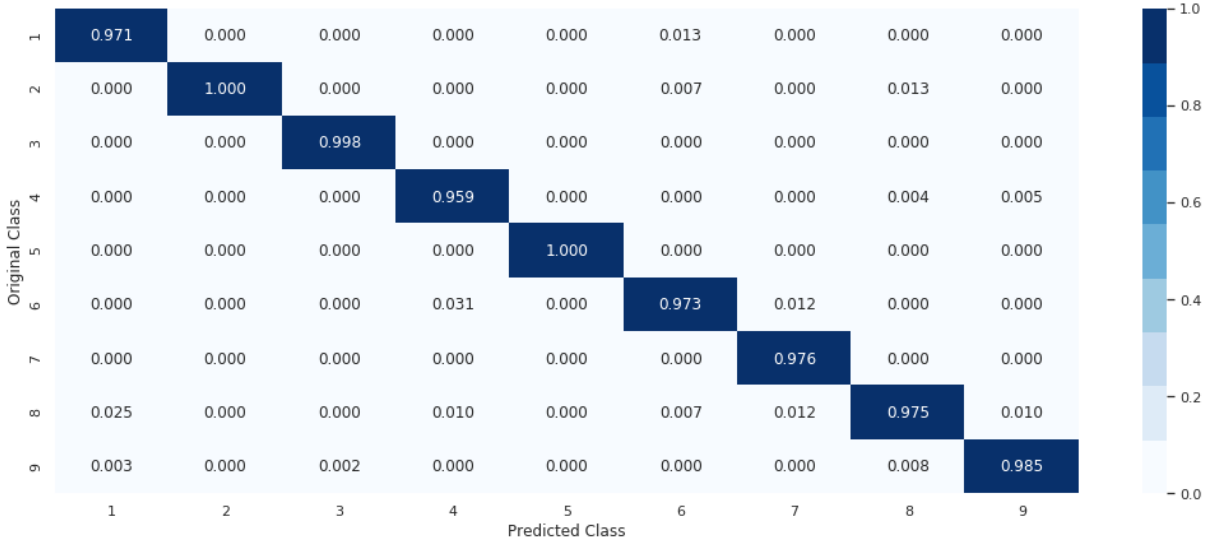
```
e': 0.15, 'colsample_bytree': 0.5}
```

In [0]:
```
def xgb_test(sub_sample, n_est, max_dpth, lr, col_sample):
    clf = XGBClassifier(n_estimators = n_est, subsample = sub_sample, m
ax_depth = max_dpth, learning_rate = lr, colsample_bytree = col_sample)
    clf.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
    sig_clf.fit(x_train, y_train)
    pred_proba_trn = sig_clf.predict_proba(x_train)
    pred_proba_tst = sig_clf.predict_proba(x_test)
    trn_err = clf.score(x_train, y_train)
    tst_err = clf.score(x_test, y_test)
    print('Log-Loss for Train-set is :', log_loss(y_train, pred_proba_t
rn))
    print('Log-Loss for Test-set is :', log_loss(y_test, pred_proba_tst
))
    print('Train Accuracy is :', trn_err)
    print('Test Accuracy is :', tst_err)
    err_metrics(y_test, sig_clf.predict(x_test))
```

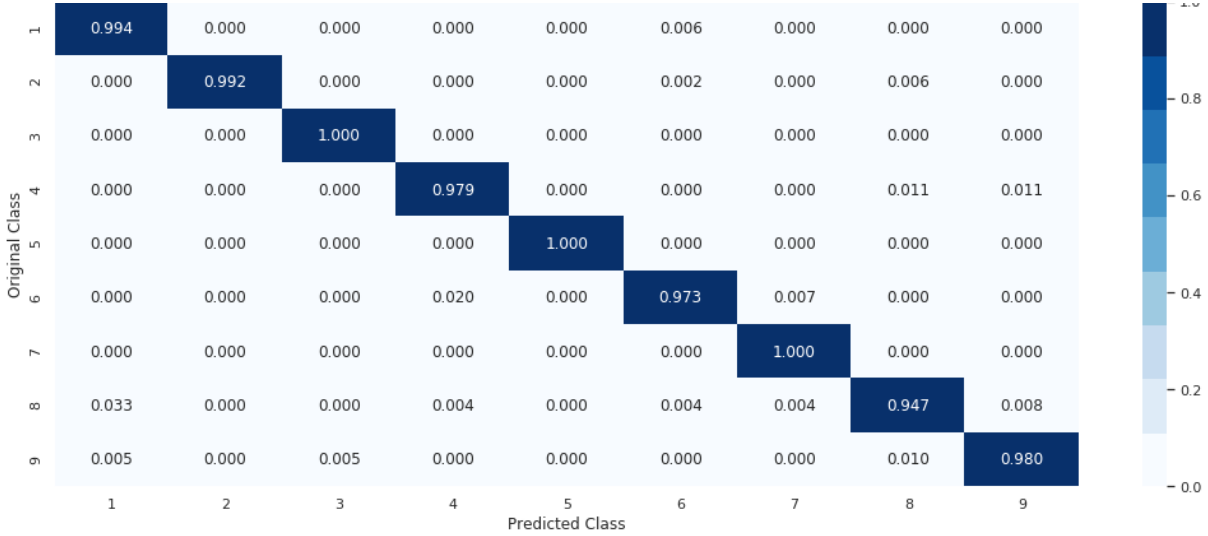
In [0]:
```
xgb_test(0.3, 500, 3, 0.15, 0.5) #Performance Metric
```

```
Log-Loss for Train-set is : 0.019985219005080344
Log-Loss for Test-set is : 0.06021760080031489
Train Accuracy is : 1.0
Test Accuracy is : 0.9875804967801288
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------



---------- RECALL-MATRIX ----------



feature extraction from ASM Files

In [0]:
```
folder_1 = 'first'
folder_2 = 'second'
folder_3 = 'third'
folder_4 = 'fourth'
folder_5 = 'fifth'
folder_6 = 'output'

for i in [folder_1, folder_2, folder_3, folder_4, folder_5, folder_6]:
```

```
    if not os.path.isdir(i):
        os.makedirs(i)

source ='../train/'
files = os.listdir('../train')
ID = result['Id'].tolist()
data = range(0,10868)
r.shuffle(data)
count = 0

for i in range(0, 10868):
    if i % 5 == 0:
        shutil.move(source + files[data[i]], 'first')
    elif i % 5 == 1:
        shutil.move(source + files[data[i]], 'second')
    elif i % 5 == 2:
        shutil.move(source + files[data[i]], 'thrid')
    elif i % 5 == 3:
        shutil.move(source + files[data[i]], 'fourth')
    elif i % 5 == 4:
        shutil.move(source + files[data[i]], 'fifth')
```

In [0]:

```
def firstprocess():

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:',
'.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'no
p', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')

    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
```

```
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")

        with codecs.open('first/'+f,encoding='cp1252',errors ='replace'
) as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        #egister only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'COD
E' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
```

```
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:',
'.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'no
p', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')

    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")

        with codecs.open('second/'+f,encoding='cp1252',errors ='replac
e') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]

                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]

                for i in range(len(opcodes)):
```

```
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1

                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'COD
E' in l):
                            registerscount[i]+=1

                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1

        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:',
'.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'no
p', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')

    for f in files:
```

```
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")

        with codecs.open('thrid/'+f,encoding='cp1252',errors ='replace'
) as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]

                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]

                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1

                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'COD
E' in l):
                            registerscount[i]+=1

                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1

        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
```

```
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fourthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:',
'.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'no
p', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')

    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")

        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replac
e') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]

                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
```

```
                line=line[1:]

                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1

                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'COD
E' in l):
                            registerscount[i]+=1

                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1

        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fifthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:',
'.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'no
p', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'cal
l', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
```

```
    files = os.listdir('fifth/')

    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")

        with codecs.open('fifth/'+f,encoding='cp1252',errors ='replace'
) as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]

                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]

                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1

                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'COD
E' in l):
                            registerscount[i]+=1

                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
```

```
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def main():

    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
    p3=Process(target=thirdprocess)
    p4=Process(target=fourthprocess)
    p5=Process(target=fifthprocess)

    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()

    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

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

In [0]:
```
asm_df = pd.read_csv("asmoutputfile.csv")
labels.columns = ['ID', 'Class']
```

```
result_asm = pd.merge(asm_df, labels, on = 'ID', how = 'left')
result_asm.head()
```

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 53 columns

In [0]:

```
asm_files = os.listdir('../asmFiles')
file_names = labels['ID'].tolist()
class_y = labels['Class'].tolist()
class_bytes = []
sizebytes = []
fnames = []

for file in asm_files:
    statinfo = os.stat('../asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the
 file name
    file = file.split('.')[0]
    if any(file == file_name for file_name in file_names):
        i = file_names.index(file)
        class_bytes.append(class_y[i])
        sizebytes.append(statinfo.st_size / (1024.0 * 1024.0))
        fnames.append(file)

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

```
   Class                    ID       size
```

```
0      2  bZ673MRJnFIlwGpj14r2  81.806677
1      4  IkdgWAqnBm803YZfuQxX   3.976359
2      9  8QzroqkiRDNMZIF69E7v   1.246348
3      2  czxjYgBb4TeOD38AhNvi  86.786336
4      1  9oCBgaw6WUrjtbSd5liq  22.152315
```

Machine Learning Models on ASM Features

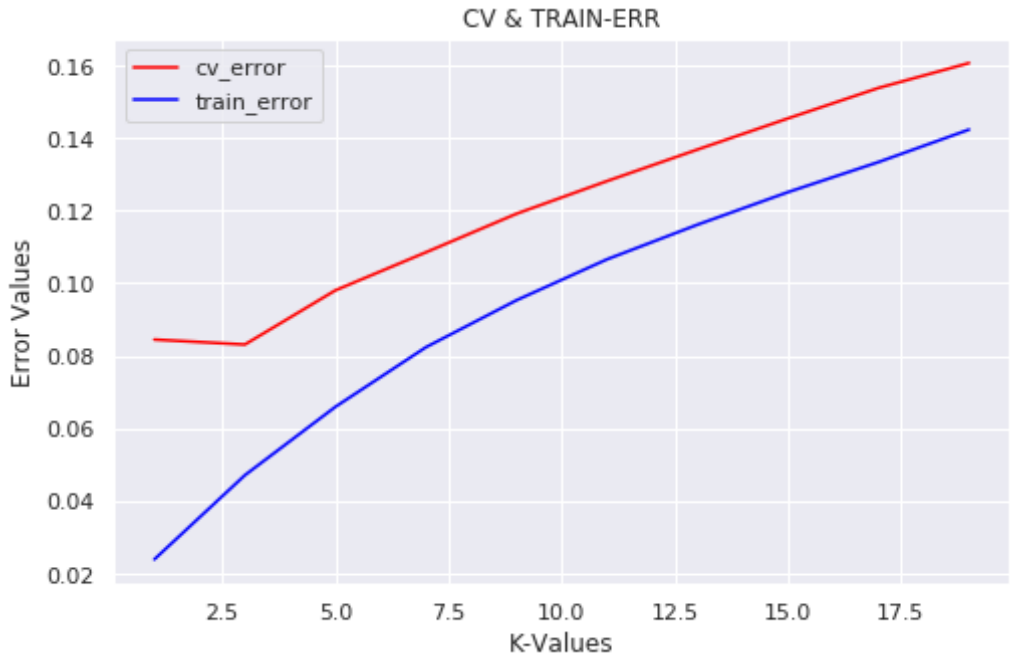
- **KNN Model**

In [0]:
```
def knn_model(operation, best_k = None):

    k_val = np.arange(1, 21, 2)
    if operation == 'Training':
        cv_err = []
        train_err = []
        for k in k_val:
            clf = KNeighborsClassifier(n_neighbors = k, n_jobs = -1)
            clf.fit(x_trn_asm, y_trn_asm)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_asm, y_trn_asm)
            pred_proba_trn = sig_clf.predict_proba(x_trn_asm)
            pred_proba_cv = sig_clf.predict_proba(x_cv_asm)
            train_err.append(log_loss(y_trn_asm, pred_proba_trn, labels
 = clf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv_asm, pred_proba_cv, labels = cl
f.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, k_val, 'K')
    else:
        clf = KNeighborsClassifier(n_neighbors = best_k, n_jobs = -1)
        clf.fit(x_train_asm, y_train_asm)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_asm, y_train_asm)
        pred_proba_trn = sig_clf.predict_proba(x_train_asm)
        pred_proba_tst = sig_clf.predict_proba(x_test_asm)
        trn_err = clf.score(x_train_asm, y_train_asm)
        tst_err = clf.score(x_test_asm, y_test_asm)
```

```
        print('Log-Loss for Train-set is :', log_loss(y_train_asm, pred
_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_asm, pred_p
roba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_asm, sig_clf.predict(x_test_asm))
```

In [0]:
```
knn_model('Training')
```

```
<matplotlib.figure.Figure at 0x7f892b2bafd0>
```

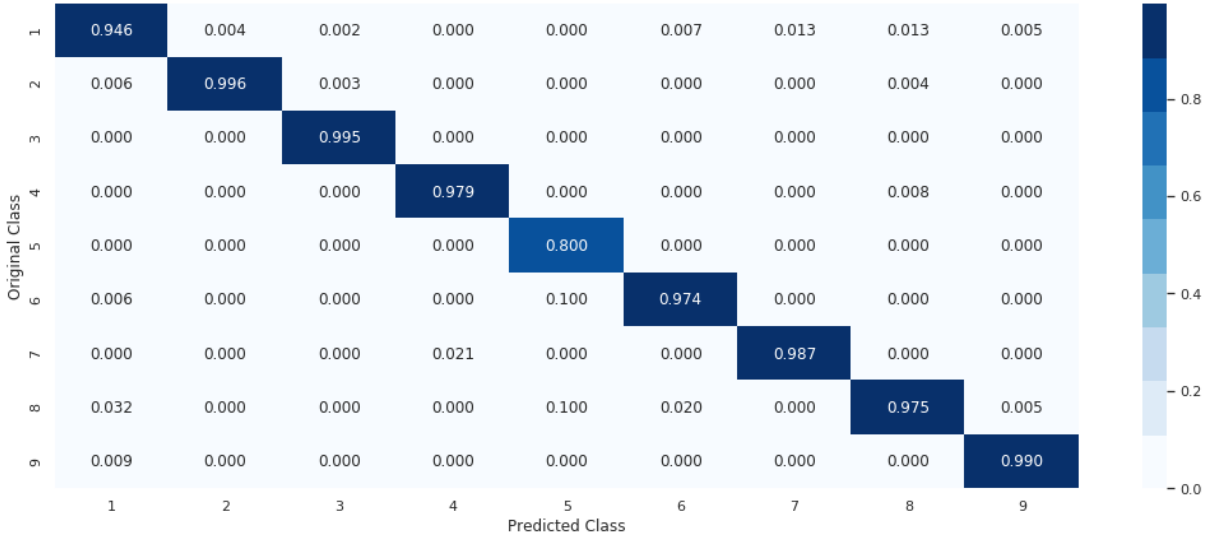


In [0]:
```
knn_model('Testing', 3)
```

```
Log-Loss for Train-set is : 0.04370532244597229
Log-Loss for Test-set is : 0.08317439880919882
Train Accuracy is : 0.9910282953761215
Test Accuracy is : 0.983440662373505
```

---------- CONFUSION-MATRIX ----------

---------- PRECISION-MATRIX ----------



---------- RECALL-MATRIX ----------

---------- RECALL-MATRIX ----------

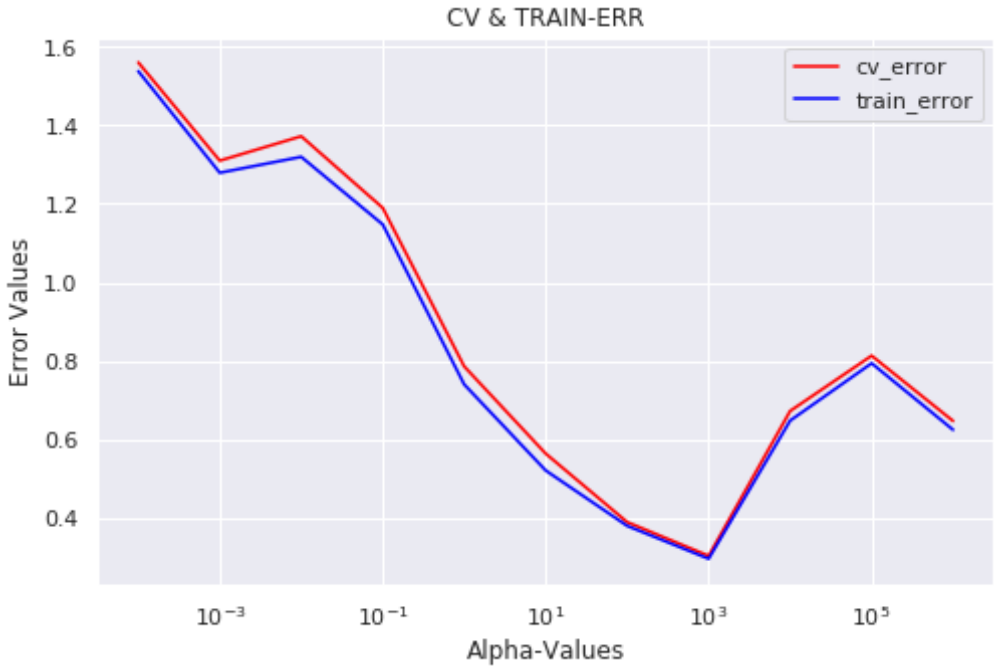
- **Logistic Regression Model**

In [0]:
```
def lr_model(operation, best_alpha = None):

    alpha = [10 ** x for x in range(-4, 7)]
    if operation == 'Training':
        cv_err = []
        train_err = []
        for a in alpha:
            clf = LogisticRegression(penalty = 'l2', C = a, class_weigh
t = 'balanced', n_jobs = -1)
            clf.fit(x_trn_asm, y_trn_asm)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_asm, y_trn_asm)
```

```
            pred_proba_trn = sig_clf.predict_proba(x_trn_asm)
            pred_proba_cv = sig_clf.predict_proba(x_cv_asm)
            train_err.append(log_loss(y_trn_asm, pred_proba_trn, labels
 = clf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv_asm, pred_proba_cv, labels = cl
f.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, alpha, 'Alpha')
    else:
        clf = LogisticRegression(penalty = 'l2', C = best_alpha, class_
weight = 'balanced', n_jobs = -1)
        clf.fit(x_train_asm, y_train_asm)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_asm, y_train_asm)
        pred_proba_trn = sig_clf.predict_proba(x_train_asm)
        pred_proba_tst = sig_clf.predict_proba(x_test_asm)
        trn_err = clf.score(x_train_asm, y_train_asm)
        tst_err = clf.score(x_test_asm, y_test_asm)
        print('Log-Loss for Train-set is :', log_loss(y_train_asm, pred
_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_asm, pred_p
roba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_asm, sig_clf.predict(x_test_asm))
```

In [0]:
```
lr_model('Training')
```

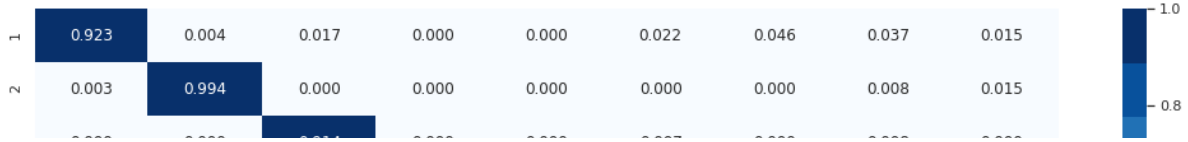
```
<matplotlib.figure.Figure at 0x7f892b27ee10>
```

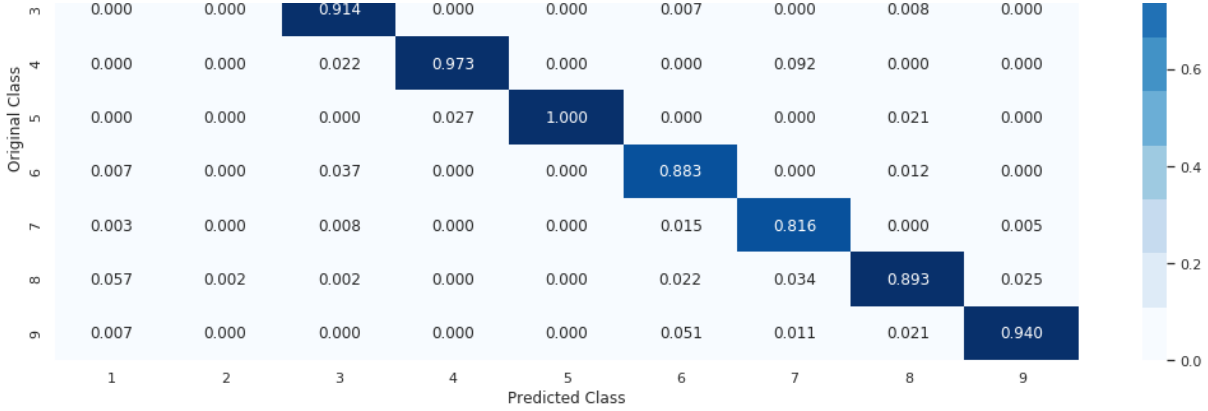


In [0]:
```
lr_model('Testing', 1000) #Performance Metric
```

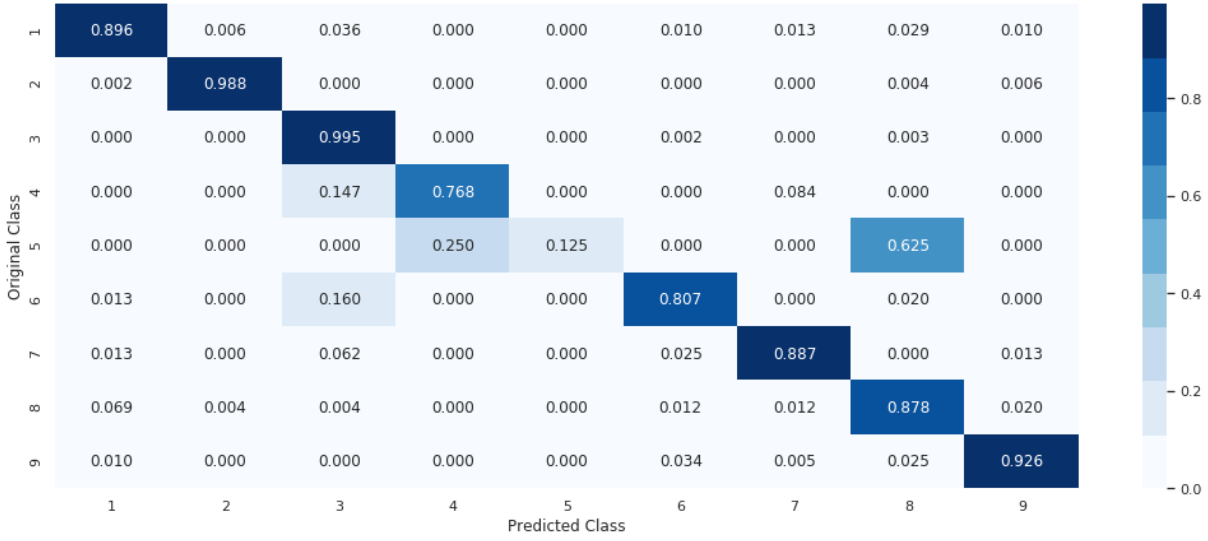
```
Log-Loss for Train-set is : 0.2989081668592061
Log-Loss for Test-set is : 0.3458853220854634
Train Accuracy is : 0.9499654934437544
Test Accuracy is : 0.9406623735050598
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------



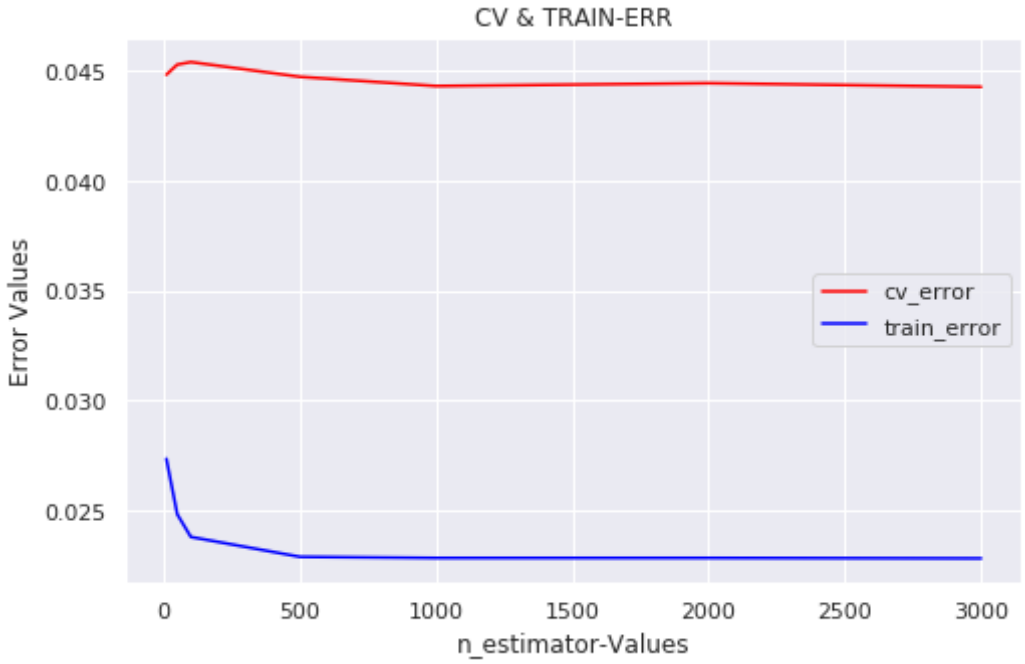
- **Decision Tree**

In [0]:
```
def dt_model(operation, best_n = None):

    n_base = [10, 50, 100, 500, 1000, 2000, 3000]
```

```
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = RandomForestClassifier(n_estimators = n, max_depth =
8, random_state = 12, n_jobs = -1)
            clf.fit(x_trn_asm, y_trn_asm)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_asm, y_trn_asm)
            pred_proba_trn = sig_clf.predict_proba(x_trn_asm)
            pred_proba_cv = sig_clf.predict_proba(x_cv_asm)
            train_err.append(log_loss(y_trn_asm, pred_proba_trn, labels
 = clf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv_asm, pred_proba_cv, labels = cl
f.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = RandomForestClassifier(n_estimators = best_n, max_depth =
 8, random_state = 12, n_jobs = -1)
        clf.fit(x_train_asm, y_train_asm)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_asm, y_train_asm)
        pred_proba_trn = sig_clf.predict_proba(x_train_asm)
        pred_proba_tst = sig_clf.predict_proba(x_test_asm)
        trn_err = clf.score(x_train_asm, y_train_asm)
        tst_err = clf.score(x_test_asm, y_test_asm)
        print('Log-Loss for Train-set is :', log_loss(y_train_asm, pred
_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_asm, pred_p
roba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_asm, sig_clf.predict(x_test_asm))
```

In [0]:
```
dt_model('Training')
```

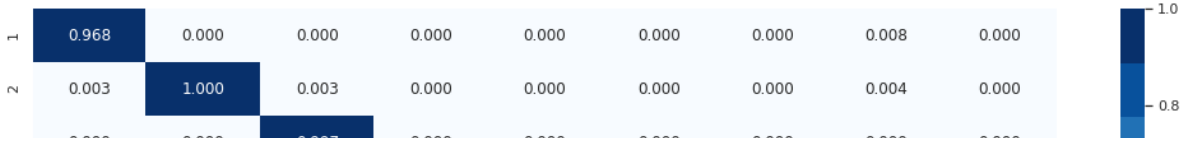
```
<matplotlib.figure.Figure at 0x7f892ae27518>
```

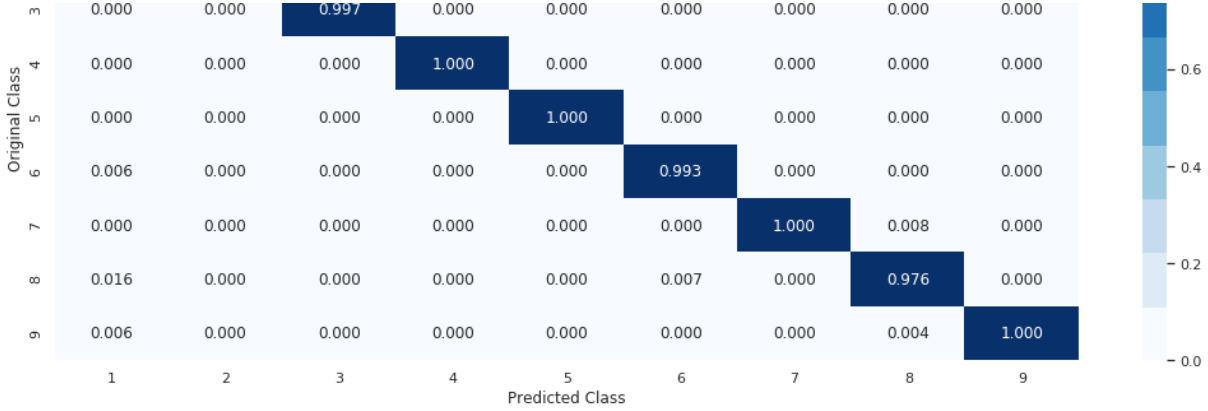


In [0]:
```
dt_model('Testing', 1000) #Performance Metric
```

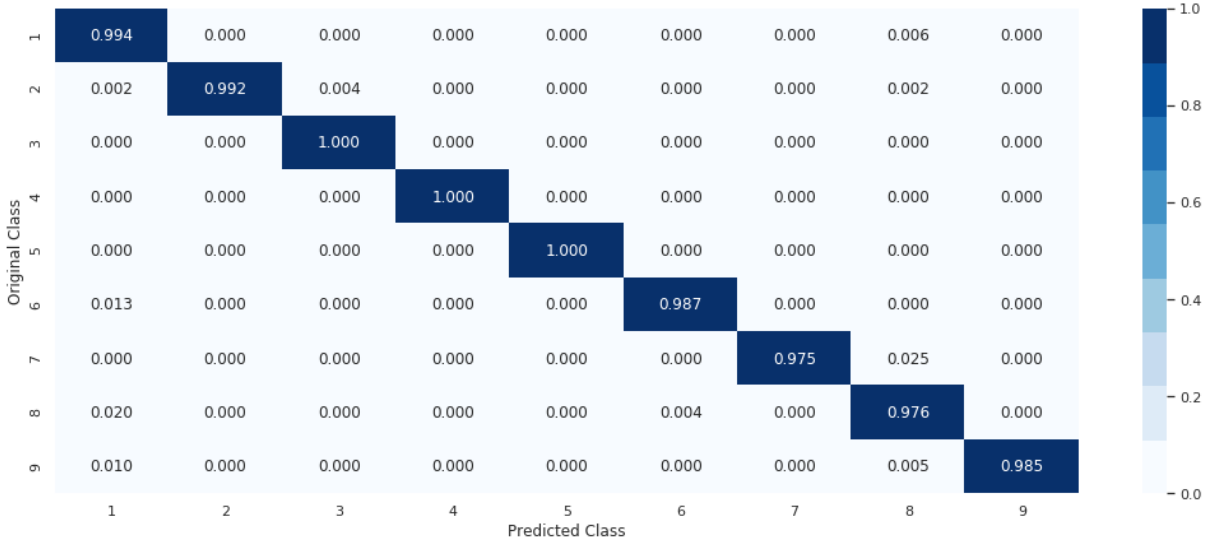
```
Log-Loss for Train-set is : 0.020847212654049
Log-Loss for Test-set is : 0.04373637694391553
Train Accuracy is : 0.9928686450425581
Test Accuracy is : 0.9889604415823368
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------



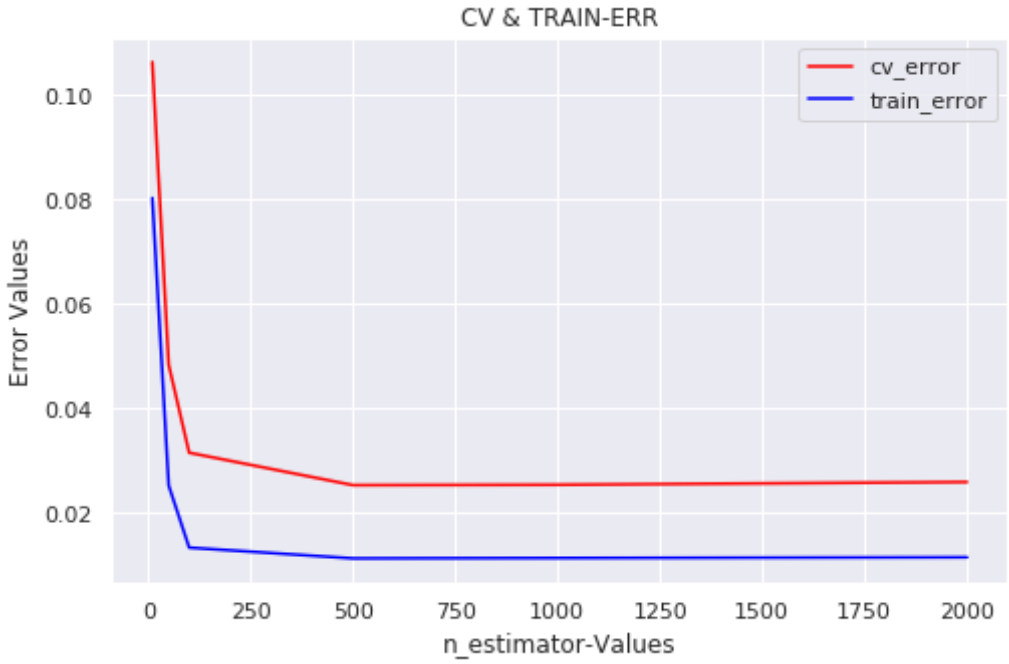
- **XGBOOST Model**

In [0]:
```
def xgb_model(operation, best_n = None):

    n_base = [10, 50, 100, 500, 1000, 2000]
```

```
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = XGBClassifier(n_estimators = n, nthread = -1)
            clf.fit(x_trn_asm, y_trn_asm)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_asm, y_trn_asm)
            pred_proba_trn = sig_clf.predict_proba(x_trn_asm)
            pred_proba_cv = sig_clf.predict_proba(x_cv_asm)
            train_err.append(log_loss(y_trn_asm, pred_proba_trn, labels
 = clf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv_asm, pred_proba_cv, labels = cl
f.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = XGBClassifier(n_estimators = best_n, nthread = -1)
        clf.fit(x_train_asm, y_train_asm)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_asm, y_train_asm)
        pred_proba_trn = sig_clf.predict_proba(x_train_asm)
        pred_proba_tst = sig_clf.predict_proba(x_test_asm)
        trn_err = clf.score(x_train_asm, y_train_asm)
        tst_err = clf.score(x_test_asm, y_test_asm)
        print('Log-Loss for Train-set is :', log_loss(y_train_asm, pred
_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_asm, pred_p
roba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_asm, sig_clf.predict(x_test_asm))
```

In [0]:
```
xgb_model('Training')
```

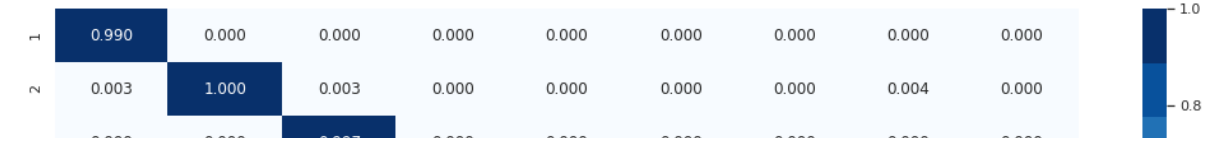
```
<matplotlib.figure.Figure at 0x7f892b2d6748>
```

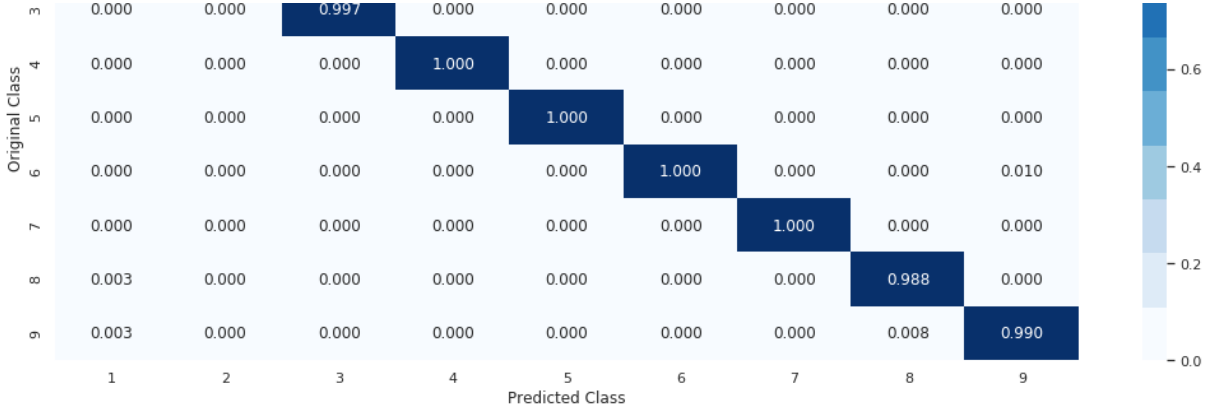


In [0]:
```
xgb_model('Testing', 500) #Performance Metric
```

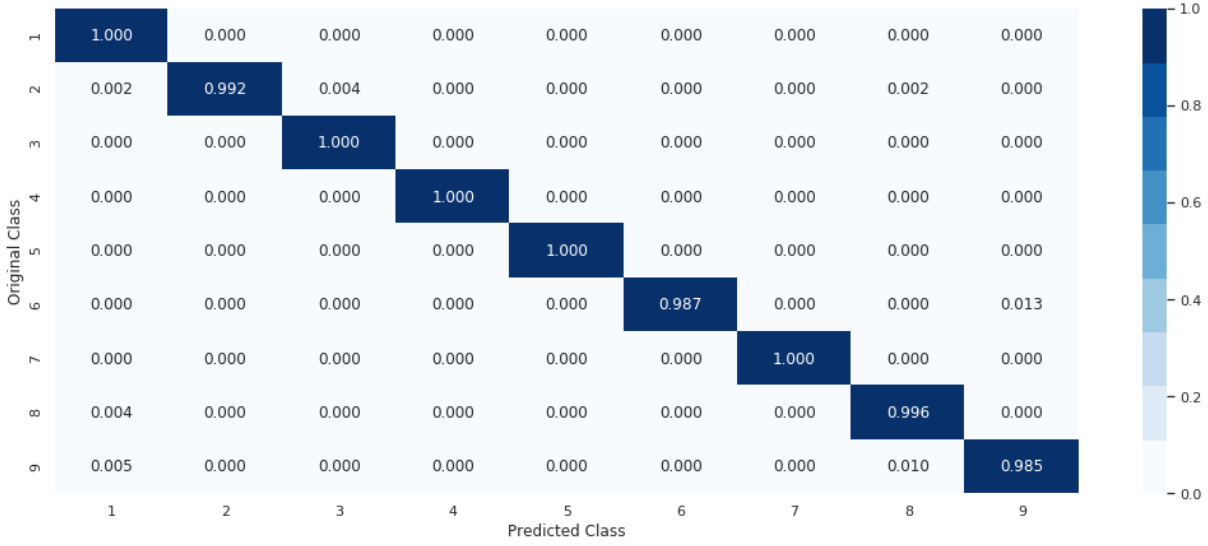
```
Log-Loss for Train-set is : 0.010012856415632923
Log-Loss for Test-set is : 0.03349910763345347
Train Accuracy is : 0.9998849781458478
Test Accuracy is : 0.9954001839926403
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------



- **hyperparameter tuning XGBOOST**

In [0]:
```
def xgb_tune():
    clf = XGBClassifier()
```

```
    prams={
        'learning_rate':[0.01, 0.03, 0.05, 0.1, 0.15, 0.2],
        'n_estimators':[100, 200, 500, 1000],
        'max_depth':[3, 5, 8],
        'colsample_bytree':[0.1, 0.3, 0.5, 1],
        'subsample':[0.1, 0.3, 0.5, 1]
    }
    random_clf = RandomizedSearchCV(clf, param_distributions = prams, v
erbose = 1, n_jobs = -1)
    random_clf.fit(x_train_asm, y_train_asm)
    print(random_clf.best_params_)
```

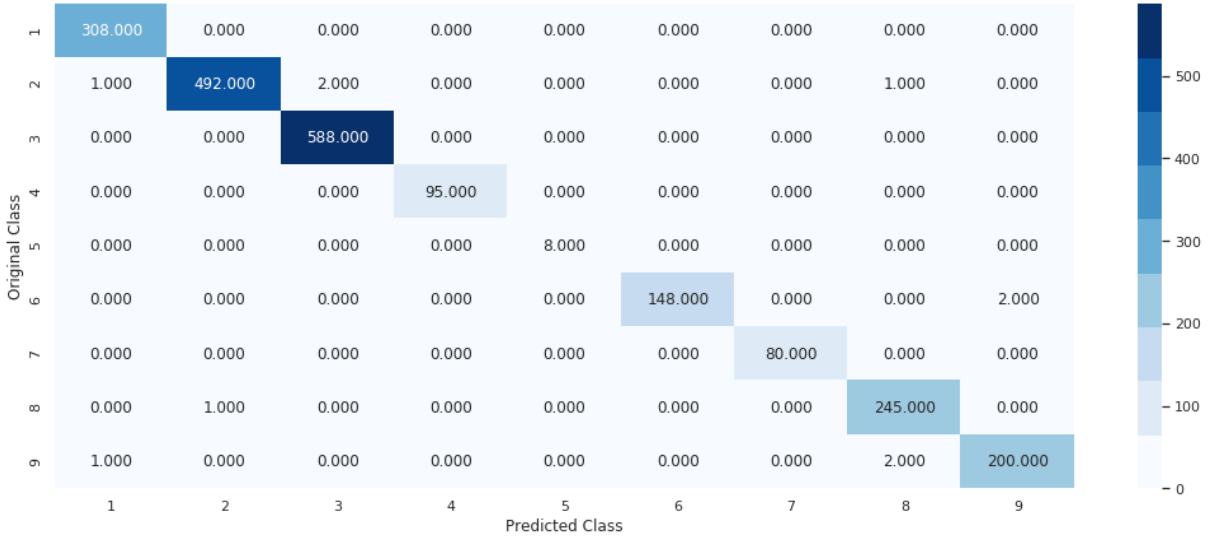
In [0]:
```
xgb_tune()
```

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

[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.6min finished

{'subsample': 1, 'n_estimators': 200, 'max_depth': 3, 'learning_rate':
0.2, 'colsample_bytree': 0.5}
```

In [0]:
```
def xgb_test(sub_sample, n_est, max_dpth, lr, col_sample):
    clf = XGBClassifier(n_estimators = n_est, subsample = sub_sample, m
ax_depth = max_dpth, learning_rate = lr, colsample_bytree = col_sample)
    clf.fit(x_train_asm, y_train_asm)
    sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
    sig_clf.fit(x_train_asm, y_train_asm)
    pred_proba_trn = sig_clf.predict_proba(x_train_asm)
    pred_proba_tst = sig_clf.predict_proba(x_test_asm)
    trn_err = clf.score(x_train_asm, y_train_asm)
    tst_err = clf.score(x_test_asm, y_test_asm)
    print('Log-Loss for Train-set is :', log_loss(y_train_asm, pred_pro
ba_trn))
    print('Log-Loss for Test-set is :', log_loss(y_test_asm, pred_proba
_tst))
    print('Train Accuracy is :', trn_err)
    print('Test Accuracy is :', tst_err)
    err_metrics(y_test_asm, sig_clf.predict(x_test_asm))
```

In [0]:
```
xgb_test(1, 200, 3, 0.2, 0.5) #Performance Metric
```

```
Log-Loss for Train-set is : 0.00903232418531879
Log-Loss for Test-set is : 0.027708849039934827
Train Accuracy is : 0.9998849781458478
Test Accuracy is : 0.9958601655933763
---------------------------------------- CONFUSION-MATRIX ------------
----------------------------
```




```
---------------------------------------- PRECISION-MATRIX ------------
----------------------------
```

---------- RECALL-MATRIX ----------

- **Machine Learning Models on ASM + Byte Features**

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

Out[0]:

	ID	0	1	2	3	4	5	
0	1IqjvX7dTxCu54M0ylBE	0.115489	0.006348	0.001659	0.004282	0.002215	0.001623	0.00235
1	CoNnhm2Se85BAHOs1kJD	0.025626	0.003013	0.000468	0.000521	0.000816	0.000353	0.00035
2	c0Hj14EGaFBbPnefqS78	0.040650	0.001045	0.000292	0.000308	0.000322	0.000329	0.00023
3	a4mHXyRFw3jUAgcdq6Op	0.039610	0.000973	0.000241	0.000294	0.000278	0.000414	0.00024
4	BcuE9gIzJoh70dAlbVqw	0.004999	0.007829	0.001776	0.001854	0.001932	0.001850	0.00181

5 rows × 260 columns

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

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 54 columns

In [0]:
```
result_x = pd.merge(result,normalize(result_asm.drop(['Class'], axis=1
)),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[0]:

	0	1	2	3	4	5	6	7	8	
0	0.115489	0.006348	0.001659	0.004282	0.002215	0.001623	0.002350	0.002921	0.002732	0.0
1	0.025626	0.003013	0.000468	0.000521	0.000816	0.000353	0.000350	0.000732	0.003161	0.0
2	0.040650	0.001045	0.000292	0.000308	0.000322	0.000329	0.000238	0.000399	0.000522	0.0
3	0.039610	0.000973	0.000241	0.000294	0.000278	0.000414	0.000248	0.000414	0.000528	0.0
4	0.004999	0.007829	0.001776	0.001854	0.001932	0.001850	0.001815	0.003038	0.002833	0.0

5 rows × 307 columns

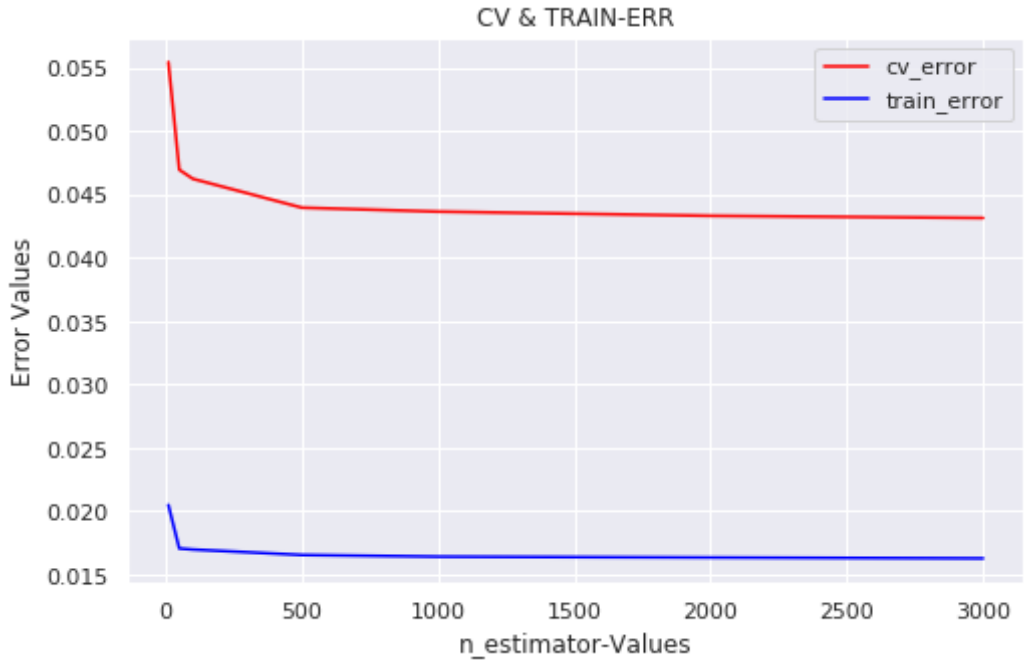
- **Decision Tree Model**

In [0]:
```
def dt_model(operation, best_n = None):

    n_base = [10, 50, 100, 500, 1000, 2000, 3000]
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = RandomForestClassifier(n_estimators = n, max_depth =
12, random_state = 12, n_jobs = -1)
            clf.fit(x_trn_total, y_trn_total)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_total, y_trn_total)
            pred_proba_trn = sig_clf.predict_proba(x_trn_total)
            pred_proba_cv = sig_clf.predict_proba(x_cv_total)
            train_err.append(log_loss(y_trn_total, pred_proba_trn, labe
ls = clf.classes_, eps = 1e-15))
```

```
            cv_err.append(log_loss(y_cv_total, pred_proba_cv, labels =
clf.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = RandomForestClassifier(n_estimators = best_n, max_depth =
 12, random_state = 12, n_jobs = -1)
        clf.fit(x_train_total, y_train_total)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_total, y_train_total)
        pred_proba_trn = sig_clf.predict_proba(x_train_total)
        pred_proba_tst = sig_clf.predict_proba(x_test_total)
        trn_err = clf.score(x_train_total, y_train_total)
        tst_err = clf.score(x_test_total, y_test_total)
        print('Log-Loss for Train-set is :', log_loss(y_train_total, pr
ed_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_total, pred
_proba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_total, sig_clf.predict(x_test_total))
```

In [0]:
```
dt_model('Training')
```

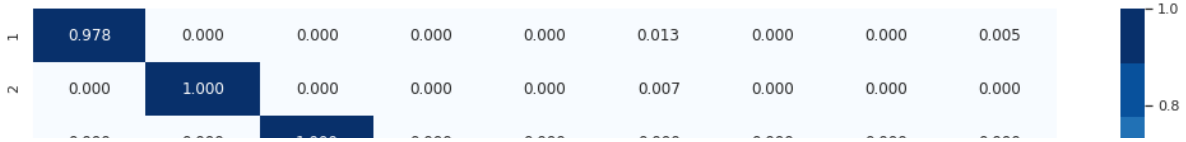
```
<matplotlib.figure.Figure at 0x7f892b8fc4a8>
```

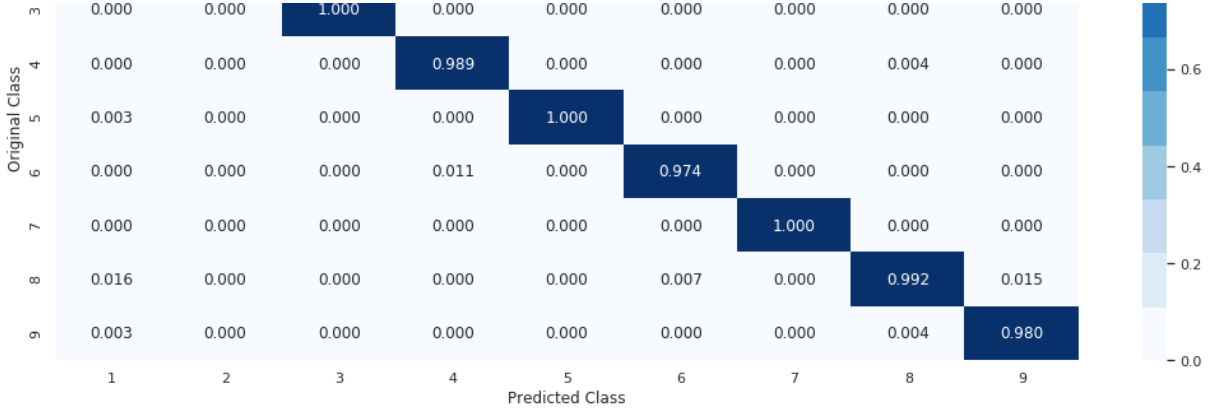


In [0]:
```
dt_model('Testing', 2000) #Performance Metric
```

```
Log-Loss for Train-set is : 0.015037301987875399
Log-Loss for Test-set is : 0.03974197515093866
Train Accuracy is : 0.999194754400092
Test Accuracy is : 0.9917203311867525
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------

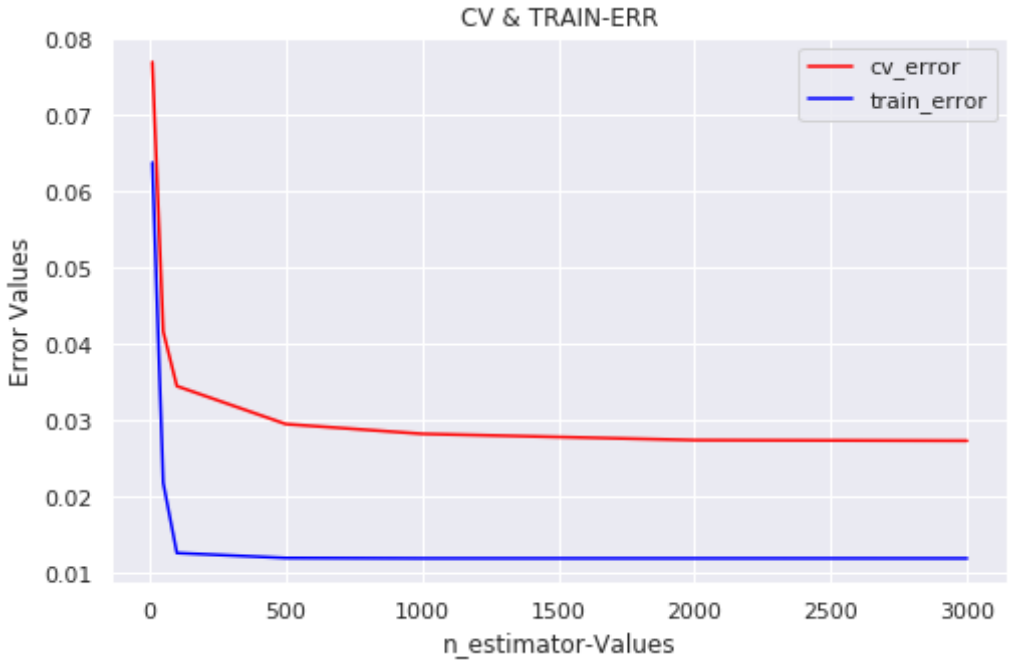
- **XGBOOST Model**

In [0]:
```
def xgb_model(operation, best_n = None):

    n_base = [10, 50, 100, 500, 1000, 2000, 3000]
```

```
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = XGBClassifier(n_estimators = n, n_jobs = -1)
            clf.fit(x_trn_total, y_trn_total)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_total, y_trn_total)
            pred_proba_trn = sig_clf.predict_proba(x_trn_total)
            pred_proba_cv = sig_clf.predict_proba(x_cv_total)
            train_err.append(log_loss(y_trn_total, pred_proba_trn, labe
ls = clf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv_total, pred_proba_cv, labels =
clf.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = XGBClassifier(n_estimators = best_n, n_jobs = -1)
        clf.fit(x_train_total, y_train_total)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_total, y_train_total)
        pred_proba_trn = sig_clf.predict_proba(x_train_total)
        pred_proba_tst = sig_clf.predict_proba(x_test_total)
        trn_err = clf.score(x_train_total, y_train_total)
        tst_err = clf.score(x_test_total, y_test_total)
        print('Log-Loss for Train-set is :', log_loss(y_train_total, pr
ed_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_total, pred
_proba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_total, sig_clf.predict(x_test_total))
```

In [0]:
```
xgb_model('Training')
```

```
<matplotlib.figure.Figure at 0x7f890eabe6a0>
```

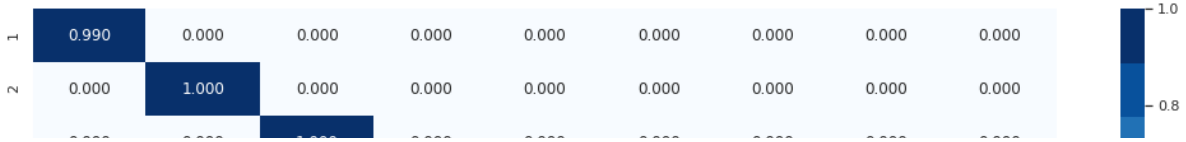


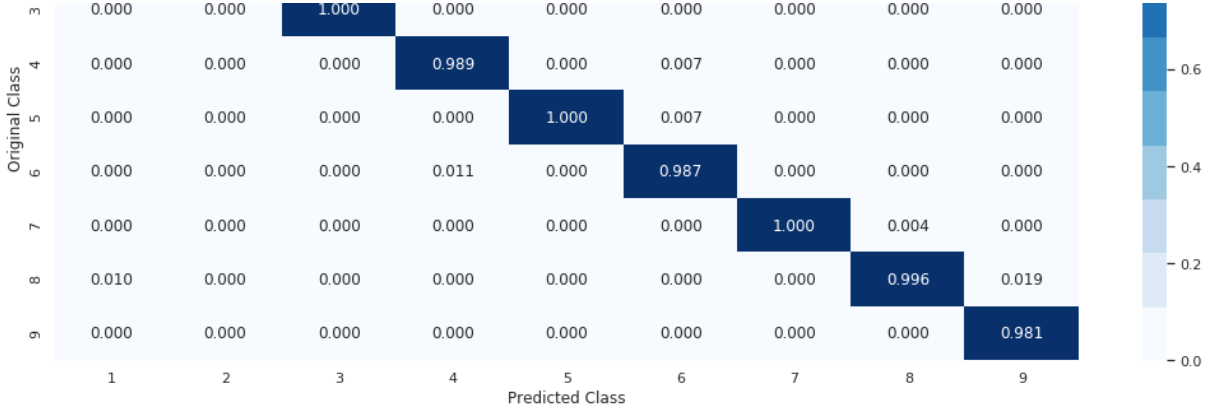
- **Performance Metric**

In [0]:
```
xgb_model('Testing', 2000) #Performance Metric
```

```
Log-Loss for Train-set is : 0.009466322267071343
Log-Loss for Test-set is : 0.031142628536659504
Train Accuracy is : 1.0
Test Accuracy is : 0.9954001839926403
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```

---------- PRECISION-MATRIX ----------





---------- RECALL-MATRIX ----------

- **XGBOOST Fine Tune**

In [0]:
```
def xgb_tune():
    clf = XGBClassifier()
```

```
    prams={
        'learning_rate':[0.01, 0.03, 0.05, 0.1, 0.15, 0.2],
        'n_estimators':[100, 200, 300, 400, 500, 1000],
        'max_depth':[3, 4, 5, 6, 7, 8],
        'colsample_bytree':[0.1, 0.3, 0.5, 1],
        'subsample':[0.1, 0.3, 0.5, 1]
    }
    random_clf = RandomizedSearchCV(clf, param_distributions = prams, v
erbose = 1, n_jobs = -1)
    random_clf.fit(x_train_total, y_train_total)
    print(random_clf.best_params_)
```

In [0]:
```
xgb_tune()
```

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

[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  4.8min finished

{'subsample': 0.5, 'n_estimators': 200, 'max_depth': 3, 'learning_rat
e': 0.1, 'colsample_bytree': 0.5}
```

In [0]:
```
def xgb_test(sub_sample, n_est, max_dpth, lr, col_sample):
    clf = XGBClassifier(n_estimators = n_est, subsample = sub_sample, m
ax_depth = max_dpth, learning_rate = lr, colsample_bytree = col_sample,
 n_jobs = -1)
    clf.fit(x_train_total, y_train_total)
    sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
    sig_clf.fit(x_train_total, y_train_total)
    pred_proba_trn = sig_clf.predict_proba(x_train_total)
    pred_proba_tst = sig_clf.predict_proba(x_test_total)
    trn_err = clf.score(x_train_total, y_train_total)
    tst_err = clf.score(x_test_total, y_test_total)
    print('Log-Loss for Train-set is :', log_loss(y_train_total, pred_p
roba_trn))
    print('Log-Loss for Test-set is :', log_loss(y_test_total, pred_pro
ba_tst))
    print('Train Accuracy is :', trn_err)
    print('Test Accuracy is :', tst_err)
    err_metrics(y_test_total, sig_clf.predict(x_test_total))
```

- **Performance Metric**

In [0]:
```
xgb_test(0.5, 200, 3, 0.1, 0.5)
```

```
Log-Loss for Train-set is : 0.010421989907717859
Log-Loss for Test-set is : 0.03485977944844367
Train Accuracy is : 1.0
Test Accuracy is : 0.9935602575896965
---------------------------------------- CONFUSION-MATRIX ------------
----------------------------
```


```
---------------------------------------- PRECISION-MATRIX ------------
----------------------------
```

---------- RECALL-MATRIX ----------

Advanced Feature Extraction

- **Bi-gram CountVectorizer on Byte File**

In [0]:
```
result_x['ID'] = result.ID
```

In [0]:
```
byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,
13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2
a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,
42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,5
9,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,
71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,87,8
8,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,
a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b
7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,
cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e
6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,
fe,ff,??"
byte_bigram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for j in range(0, len(byte_vocab.split(','))):
        byte_bigram_vocab.append(v + ' ' +byte_vocab.split(',')[j])
len(byte_bigram_vocab)
```

Out[0]:
```
66049
```

In [0]:
```
byte_bigram_vocab[:5]
```

Out[0]:
```
['00 00', '00 01', '00 02', '00 03', '00 04']
```

In [0]:
```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
```

```
vect = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=by
te_bigram_vocab)
byte_bigram_vect = scipy.sparse.csr_matrix((10868, 66049))
for i, file in tqdm(enumerate(os.listdir('../byteFiles'))):
    f = open('../byteFiles/' + file)
    a[i, :]+= scipy.sparse.csr_matrix(vect.fit_transform([f.read().repl
ace('\n', ' ').lower()]))
    f.close()
```

```
10868it [31:52:28, 20.40s/it]
```

In [0]:
```
byte_bigram_vect
```

Out[0]:
```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'
        with 502081976 stored elements in Compressed Sparse Row format>
```

In [0]:
```
scipy.sparse.save_npz('byte_bigram.npz', byte_bigram_vect)
```

In [0]:
```
from sklearn.preprocessing import normalize
byte_bigram_vect = normalize(scipy.sparse.load_npz('byte_bigram.npz'),
axis = 0)
```

- **N-Gram(2-Gram, 3-Gram, 4-Gram) Opcode Vectorization**

In [0]:
```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop',
'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
```

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

Out[0]:
```
676
```

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

Out[0]:
```
17576
```

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

Out[0]:
```
456976
```

In [0]:
```
def opcode_collect():
    op_file = open("../opcode_file.txt", "w+")
    for asmfile in os.listdir('../asmFiles'):
        opcode_str = ""
        with codecs.open('../asmFiles/' + asmfile, encoding='cp1252', e
rrors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
opcode_collect()
```

In [0]:
```
vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_opcode_bigr
am)
```

```
opcode_bi_vect = scipy.sparse.csr_matrix((10868, len(asm_opcode_bigram
)))
raw_opcode = open('../opcode_file.txt').read().split('\n')

for indx in range(10868):
    opcode_bi_vect[indx, :] += scipy.sparse.csr_matrix(vect.transform([
raw_opcode[indx]]))
```

In [0]:
```
opcode_bi_vect
```

Out[0]:
```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'
        with 1877309 stored elements in Compressed Sparse Row format>
```

In [0]:
```
scipy.sparse.save_npz('opcode_bigram.npz', opcode_bi_vect)
```

In [0]:
```
opcode_bi_vect = scipy.sparse.load_npz('opcode_bigram.npz')
```

In [0]:
```
vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_opcode_trig
ram)
opcode_tri_vect = scipy.sparse.csr_matrix((10868, len(asm_opcode_trigra
m)))

for indx in range(10868):
    opcode_tri_vect[indx, :] += scipy.sparse.csr_matrix(vect.transform
([raw_opcode[indx]]))
```

In [0]:
```
opcode_tri_vect
```

Out[0]:
```
<10868x17576 sparse matrix of type '<class 'numpy.float64'>'
        with 7332672 stored elements in Compressed Sparse Row format>
```

In [0]:
```
scipy.sparse.save_npz('opcode_trigram.npz', opcode_tri_vect)
```

In [0]:
```
opcode_tri_vect = scipy.sparse.load_npz('opcode_trigram.npz')
```

In [0]:
```
vect = CountVectorizer(ngram_range=(4, 4), vocabulary = asm_opcode_tetr
```

```
agram)
opcode_tetra_vect = scipy.sparse.csr_matrix((10868, len(asm_opcode_tetr
agram)))

for indx in range(10868):
    opcode_tetra_vect[indx, :] += scipy.sparse.csr_matrix(vect.transfor
m([raw_opcode[indx]]))
```

In [0]:
```
opcode_tetra_vect
```

Out[0]:
```
<10868x456976 sparse matrix of type '<class 'numpy.float64'>'
        with 16605229 stored elements in Compressed Sparse Row format>
```

In [0]:
```
scipy.sparse.save_npz('opcode_tetragram.npz', opcode_tetra_vect)
```

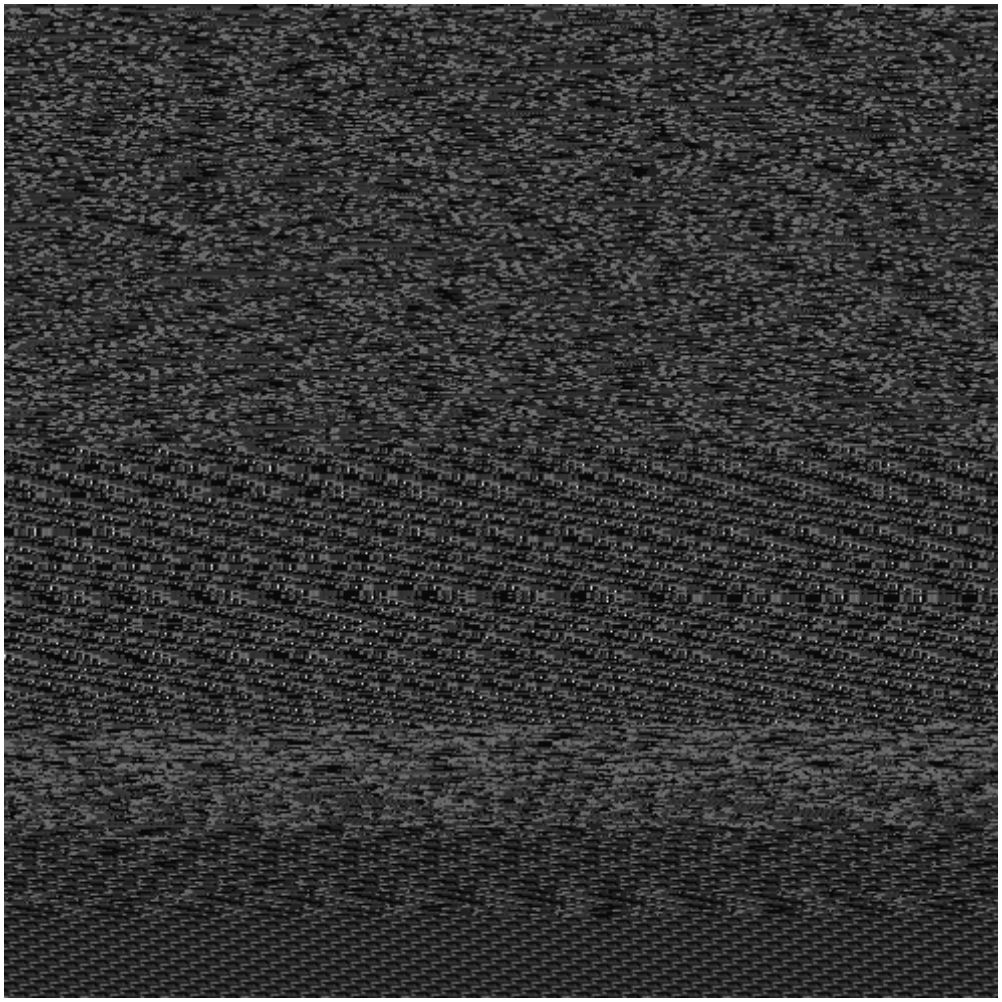
In [0]:
```
opcode_tetra_vect = scipy.sparse.load_npz('opcode_tetragram.npz')
```

- **Image Feature Extraction From ASM Files**

In [0]:
```
import array
def collect_img_asm():
    #pix_file = open("../pixels.txt", "w+")
    for asmfile in os.listdir("../asmFiles"):
        file_name = asmfile.split('.')[0]
        file = codecs.open("../asmFiles/" + asmfile, 'rb')
        file_len = os.path.getsize("../asmFiles/" + asmfile)
        width = int(file_len ** 0.5)
        rem = int(file_len / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        scipy.misc.imsave('../asm_image/' + file_name + '.png',reshaped
)
collect_img_asm()
```

In [0]:
```
from IPython.display import Image
Image(filename='../asm_image/deTXH9Zau7qmM0yfYsRS.png')
```

Out[0]:



- **First 800 Image Pixels**

In [0]:
```
import cv2
image_features = np.zeros((10868, 800))
```

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

In [0]:
```
from sklearn.preprocessing import normalize

img_features_name = []
for i in range(800):
    img_features_name.append('pix' + str(i))
img_df = pd.DataFrame(normalize(image_features, axis = 0), columns = im
g_features_name)
```

In [0]:
```
img_df['ID'] = result.ID
```

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

Out[0]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	
0	0.010269	0.010269	0.010269	0.008034	0.008034	0.008034	0.008320	0.008320	0.008320	0.0
1	0.010269	0.010269	0.010269	0.008034	0.008034	0.008034	0.008320	0.008320	0.008320	0.0
2	0.006560	0.006560	0.006560	0.013506	0.013506	0.013506	0.012928	0.012928	0.012928	0.0
3	0.010269	0.010269	0.010269	0.008034	0.008034	0.008034	0.008320	0.008320	0.008320	0.0
4	0.010269	0.010269	0.010269	0.008034	0.008034	0.008034	0.008320	0.008320	0.008320	0.0

5 rows × 801 columns

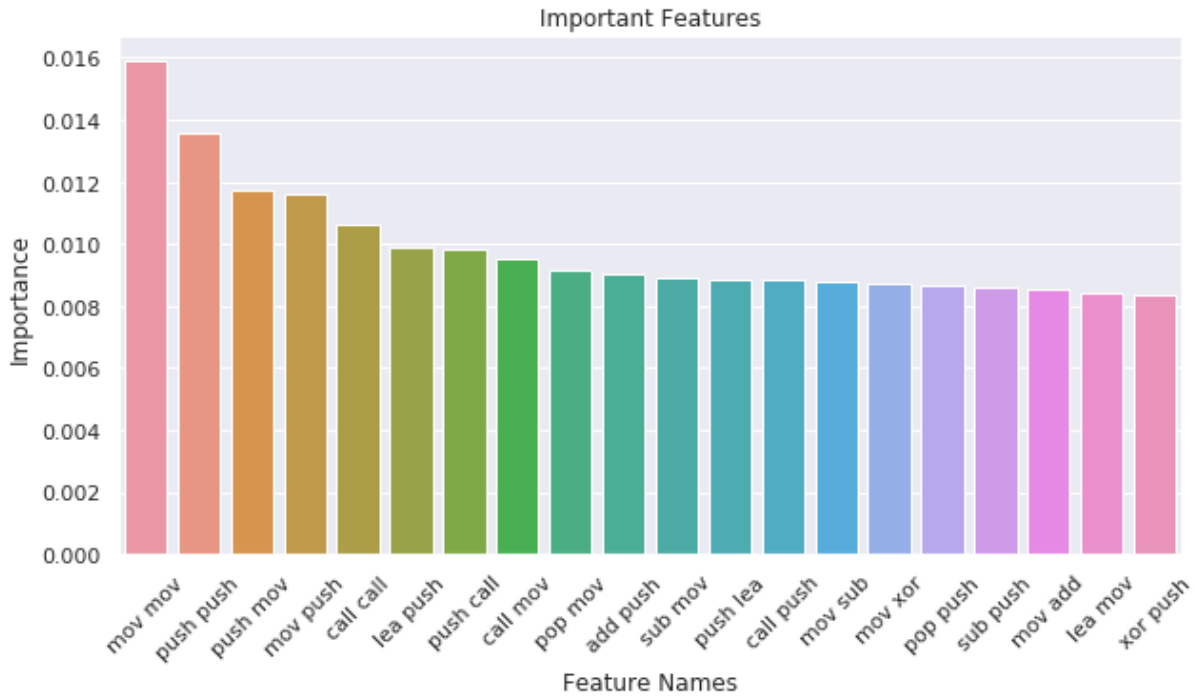
- **Important Feature Selection Using Random Forest**

In [0]:
```
from sklearn.ensemble import RandomForestClassifier

def imp_features(data, features, keep):
```

```
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])

    sns.set()
    plt.figure(figsize = (10, 5))
    ax = sns.barplot(x = imp_feature_name, y = imp_value)
    ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
    sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
    plt.title('Important Features')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')

    return imp_feature_indx[:keep]
```

- **Important Feature Among Opcode Bi-Gram**

In [0]:
```
op_bi_indxes = imp_features(normalize(opcode_bi_vect, axis = 0), asm_op
code_bigram, 300)
```



In [0]:
```
op_bi_df = pd.SparseDataFrame(normalize(opcode_bi_vect, axis = 0), colu
mns = asm_opcode_bigram)
for col in op_bi_df.columns:
    if col not in np.take(asm_opcode_bigram, op_bi_indxes):
        op_bi_df.drop(col, axis = 1, inplace = True)
```

In [0]:
```
op_bi_df.to_dense().to_csv('op_bi_filtered.csv')
```

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

In [0]:
```
op_bi_df['ID'] = result.ID
op_bi_df.head()
```

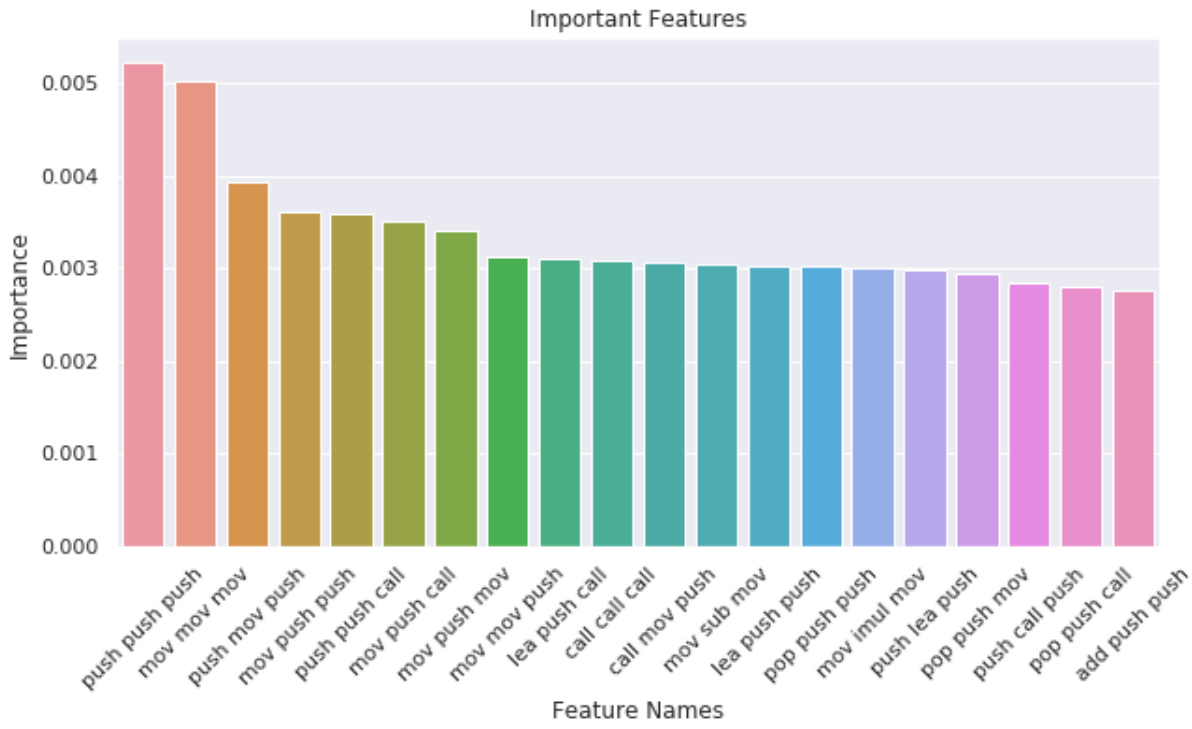
Out[0]:

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp retn	jmp sub	jmp inc	jmp dec	jm
0	0.030802	0.003042	0.014499	0.065875	0.003212	0.014054	0.041021	0.025327	0.039309	0.0
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
2	0.000000	0.000710	0.000280	0.000000	0.000140	0.000000	0.000000	0.001809	0.000000	0.0
3	0.009327	0.046123	0.054355	0.005614	0.045391	0.062541	0.038784	0.031659	0.027896	0.0
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0

5 rows × 301 columns

- **Important Feature Among Opcode 3-Gram**

In [0]:
```
op_tri_indxes = imp_features(normalize(opcode_tri_vect, axis = 0), asm_
opcode_trigram, 2000)
```



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

In [0]:
```
op_tri_df.to_dense().to_csv('op_tri_filtered.csv')
```

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

In [0]:
```
op_tri_df['ID'] = result.ID
op_tri_df.head()
```

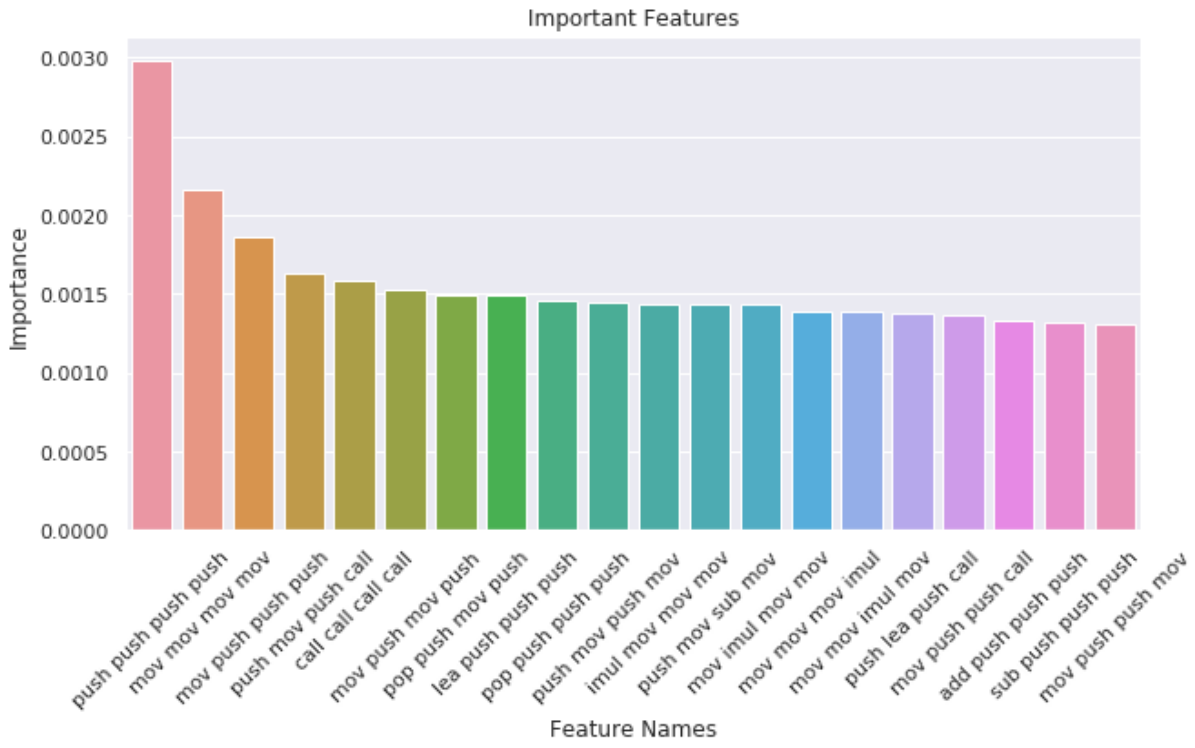
Out[0]:

	add add add	add add cmp	add add dec	add add inc	add add jmp	add add jz	add add lea	add add mov	add add or	ad
0	0.000000	0.000000	0.000000	0.000000	0.022701	0.000000	0.000000	0.000000	0.000000	0.0
1	0.000300	0.001274	0.000000	0.000000	0.000000	0.004906	0.005920	0.000285	0.002670	0.0
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
3	0.018811	0.064963	0.003599	0.003524	0.040357	0.034343	0.069064	0.057932	0.037383	0.0
4	0.000100	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0

5 rows × 2001 columns

- **Important Feature Among Opcode 4-Gram**

In [0]:
```
op_tetra_indxes = imp_features(normalize(opcode_tetra_vect, axis = 0),
asm_opcode_tetragram, 5000)
```



In [0]:
```
op_tetra_df = pd.SparseDataFrame(normalize(opcode_tetra_vect, axis = 0
), columns = asm_opcode_tetragram)
op_tetra_df = op_tetra_df.loc[:, np.intersect1d(op_tetra_df.columns, np
.take(asm_opcode_tetragram, op_tetra_indxes))]
```

In [0]:
```
op_tetra_df.to_dense().to_csv('op_tetra_filtered.csv')
```

In [0]:
```
op_tetra_df = pd.read_csv('op_tetra_filtered.csv').drop('Unnamed: 0', a
xis = 1).fillna(0)
```

In [0]:
```
op_tetra_df['ID'] = result.ID
op_tetra_df.head()
```

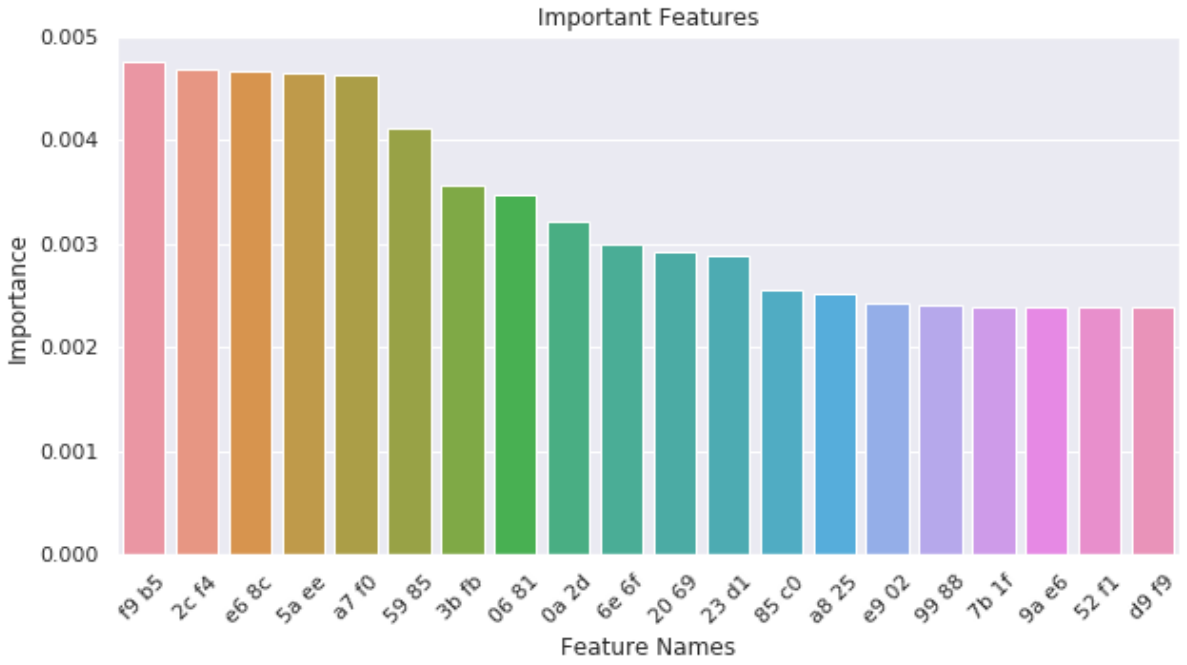
Out[0]:

	add add add add	add add add cmp	add add add dec	add add add jmp	add add add mov	add add add or	add add add push	add add add retn	add add add sub	add add cmp call	...
0	0.000000	0.000000	0.00000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000000	...
1	0.000000	0.000000	0.00000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000000	...
2	0.000000	0.000000	0.00000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000000	...
3	0.008631	0.061381	0.00703	0.022771	0.051918	0.0	0.088517	0.0	0.024271	0.029761	...
4	0.000000	0.000000	0.00000	0.000000	0.000000	0.0	0.000000	0.0	0.001348	0.000000	...

5 rows × 5001 columns

- **Important feature among byte Bi-Gram**

In [0]:
```
byte_bi_indxes = imp_features(normalize(byte_bigram_vect, axis = 0), byte_bigram_vocab, 5000)
```



In [0]:
```
np.save('byte_bi_indx', byte_bi_indxes)
```

In [0]:
```
byte_bi_indxes = np.load('byte_bi_indx.npy')
```

In [0]:
```
top_byte_bi = np.zeros((10868, 1))
for i in byte_bi_indxes:
    sliced = byte_bigram_vect[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

In [0]:
```
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns = np.take(byte_big
ram_vocab, byte_bi_indxes))
```

In [0]:
```
byte_bi_df.to_dense().to_csv('byte_bi_filtered.csv')
```

In [0]:
```
byte_bi_df = pd.read_csv('byte_bi_filtered.csv').drop('Unnamed: 0', axi
s = 1).fillna(0)
```

In [0]:
```
byte_bi_df['ID'] = result.ID
byte_bi_df.head()
```

Out[0]:

	f9 b5	2c f4	e6 8c	5a ee	a7 f0	59 85	3b fb	06 81	0a 2d	
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001424	0.000000	0.000000	0.0
1	0.000000	0.000063	0.000951	0.000014	0.001614	0.000267	0.009615	0.000007	0.000017	0.0
2	0.001846	0.000063	0.000951	0.000028	0.000000	0.000053	0.000712	0.000013	0.000033	0.0
3	0.000000	0.000063	0.000951	0.000014	0.000000	0.000000	0.000356	0.000007	0.000000	0.0
4	0.012920	0.000564	0.014263	0.000234	0.017759	0.000694	0.004986	0.000099	0.000116	0.0

5 rows × 5001 columns

- **Byte Features + ASM Features + Byte Bi-Gram + Opcode Bi-Gram + Opcode 3-Gram + Opcode 4-Gram + Pixel Intensity from ASM Image**

In [0]:
```
final_data = pd.concat([result_x, op_bi_df, op_tri_df, op_tetra_df, byte_bi_df, img_df], axis = 1, join = 'inner')
```

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

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

Out[0]:

	0	1	2	3	4	5	6	7	8	
0	0.115489	0.006348	0.001659	0.004282	0.002215	0.001623	0.002350	0.002921	0.002732	0.0
1	0.025626	0.003013	0.000468	0.000521	0.000816	0.000353	0.000350	0.000732	0.003161	0.0
2	0.040650	0.001045	0.000292	0.000308	0.000322	0.000329	0.000238	0.000399	0.000522	0.0
3	0.039610	0.000973	0.000241	0.000294	0.000278	0.000414	0.000248	0.000414	0.000528	0.0

	0	1	2	3	4	5	6	7	8	
4	0.004999	0.007829	0.001776	0.001854	0.001932	0.001850	0.001815	0.003038	0.002833	0.0

5 rows × 13407 columns

In [0]:
```
final_data.to_csv('final_data.csv')
```

In [0]:
```
final_data = pd.read_csv('final_data.csv')
```

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

Out[0]:

	Unnamed: 0	0	1	2	3	4	5	6	7	
0	0	0.115489	0.006348	0.001659	0.004282	0.002215	0.001623	0.002350	0.002921	0.
1	1	0.025626	0.003013	0.000468	0.000521	0.000816	0.000353	0.000350	0.000732	0.
2	2	0.040650	0.001045	0.000292	0.000308	0.000322	0.000329	0.000238	0.000399	0.
3	3	0.039610	0.000973	0.000241	0.000294	0.000278	0.000414	0.000248	0.000414	0.
4	4	0.004999	0.007829	0.001776	0.001854	0.001932	0.001850	0.001815	0.003038	0.

5 rows × 13408 columns

In [0]:
```
x_train_final, x_test_final, y_train_final, y_test_final = train_test_s
plit(final_data, result_y, stratify = result_y, test_size = 0.20)
x_trn_final, x_cv_final, y_trn_final, y_cv_final = train_test_split(x_t
rain_final, y_train_final, stratify = y_train_final, test_size = 0.20)
```

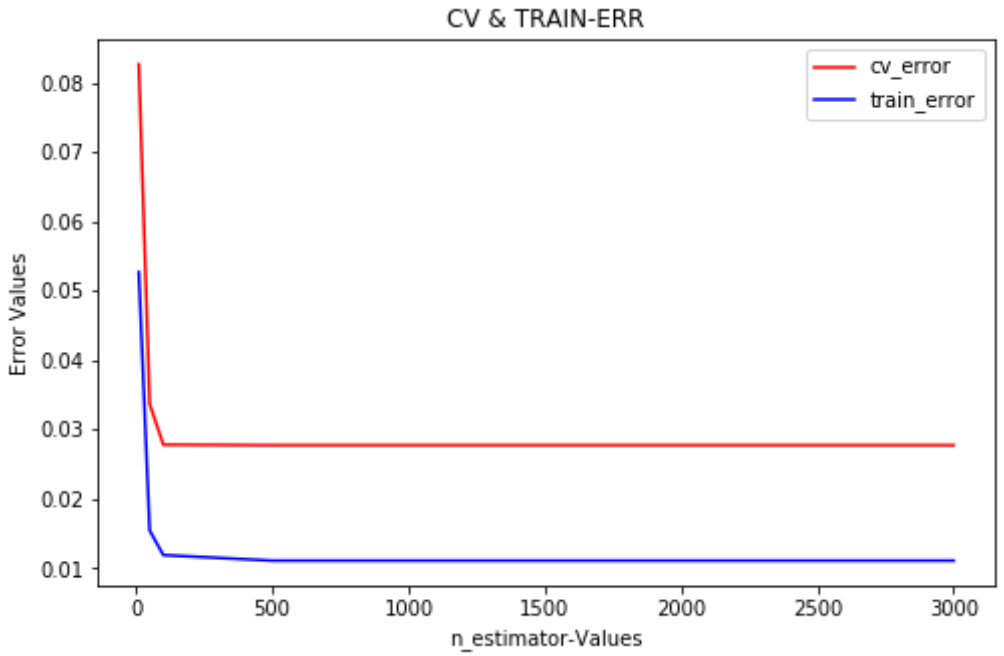
- **Machine Learning Models on ASM Features + Byte Features + Advanced Features**

- **XGBOOST Model**

In [0]:
```
def xgb_model(operation, best_n = None):

    n_base = [10, 50, 100, 500, 1000, 2000, 3000]
    if operation == 'Training':
        cv_err = []
        train_err = []
        for n in n_base:
            clf = XGBClassifier(n_estimators = n, n_jobs = -1)
            clf.fit(x_trn_final, y_trn_final)
            sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
            sig_clf.fit(x_trn_final, y_trn_final)
            pred_proba_trn = sig_clf.predict_proba(x_trn_final)
            pred_proba_cv = sig_clf.predict_proba(x_cv_final)
            train_err.append(log_loss(y_trn_final, pred_proba_trn, labe
ls = clf.classes_, eps = 1e-15))
            cv_err.append(log_loss(y_cv_final, pred_proba_cv, labels =
clf.classes_, eps = 1e-15))
        err_compare(train_err, cv_err, n_base, 'n_estimator')
    else:
        clf = XGBClassifier(n_estimators = best_n, n_jobs = -1)
        clf.fit(x_train_final, y_train_final)
        sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
        sig_clf.fit(x_train_final, y_train_final)
        pred_proba_trn = sig_clf.predict_proba(x_train_final)
        pred_proba_tst = sig_clf.predict_proba(x_test_final)
        trn_err = clf.score(x_train_final, y_train_final)
        tst_err = clf.score(x_test_final, y_test_final)
        print('Log-Loss for Train-set is :', log_loss(y_train_final, pr
ed_proba_trn))
        print('Log-Loss for Test-set is :', log_loss(y_test_final, pred
_proba_tst))
        print('Train Accuracy is :', trn_err)
        print('Test Accuracy is :', tst_err)
        err_metrics(y_test_final, sig_clf.predict(x_test_final))
```

In [0]:
```
xgb_model('Training')
```

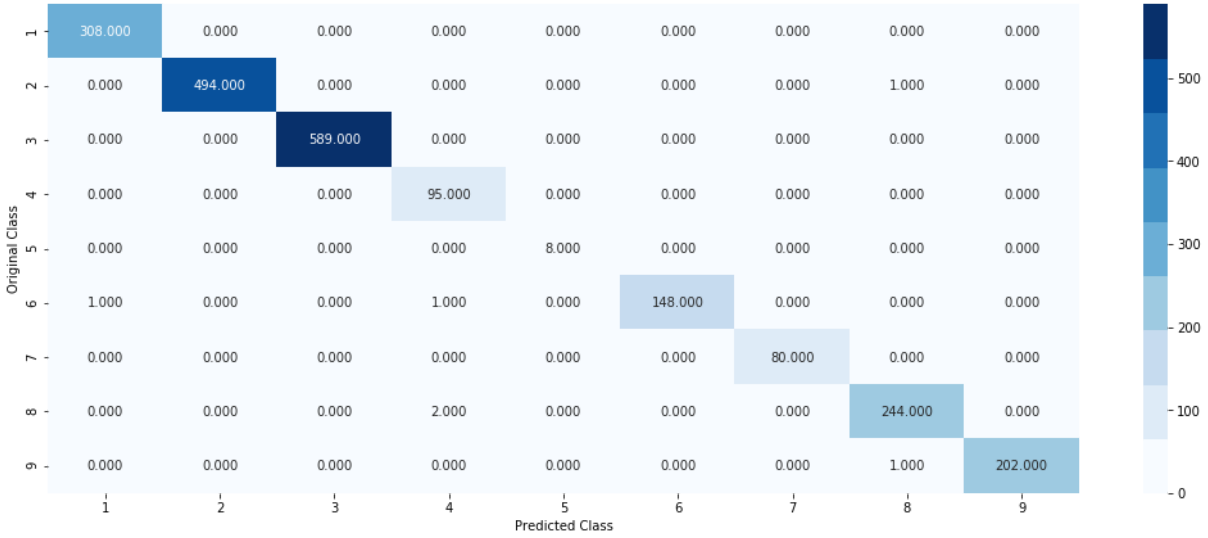
```
<matplotlib.figure.Figure at 0x7fec8be68f98>
```



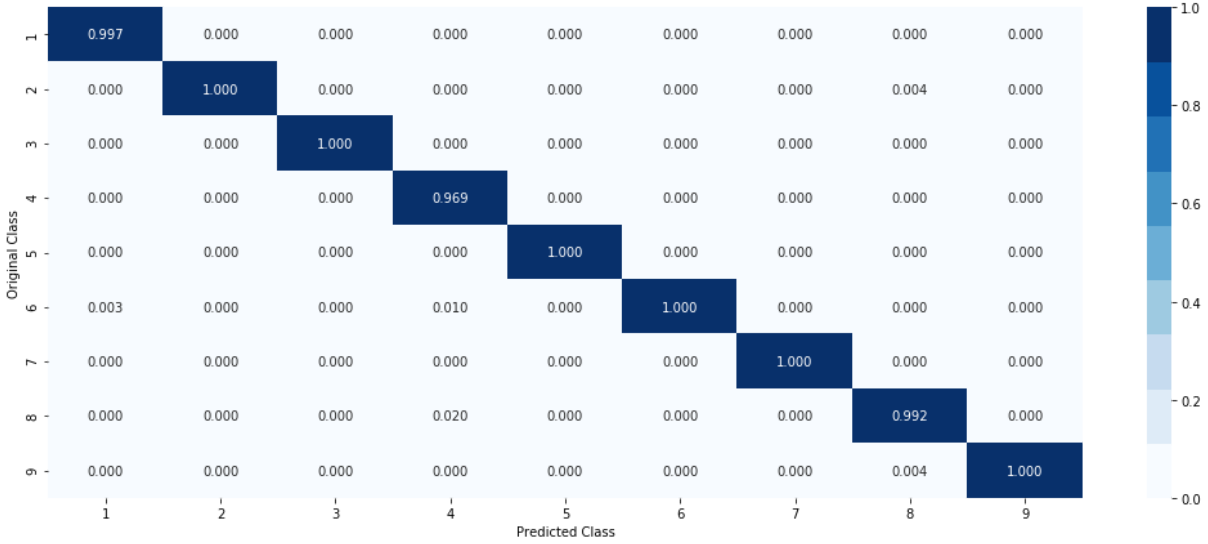
In [0]:
```
xgb_model('Testing', 2000)
```

```
/home/sradheya/anaconda3/lib/python3.6/site-packages/sklearn/preprocess
ing/label.py:151: DeprecationWarning: The truth value of an empty array
is ambiguous. Returning False, but in future this will result in an err
or. Use `array.size > 0` to check that an array is not empty.
  if diff:
/home/sradheya/anaconda3/lib/python3.6/site-packages/sklearn/preprocess
ing/label.py:151: DeprecationWarning: The truth value of an empty array
is ambiguous. Returning False, but in future this will result in an err
or. Use `array.size > 0` to check that an array is not empty.
  if diff:
```

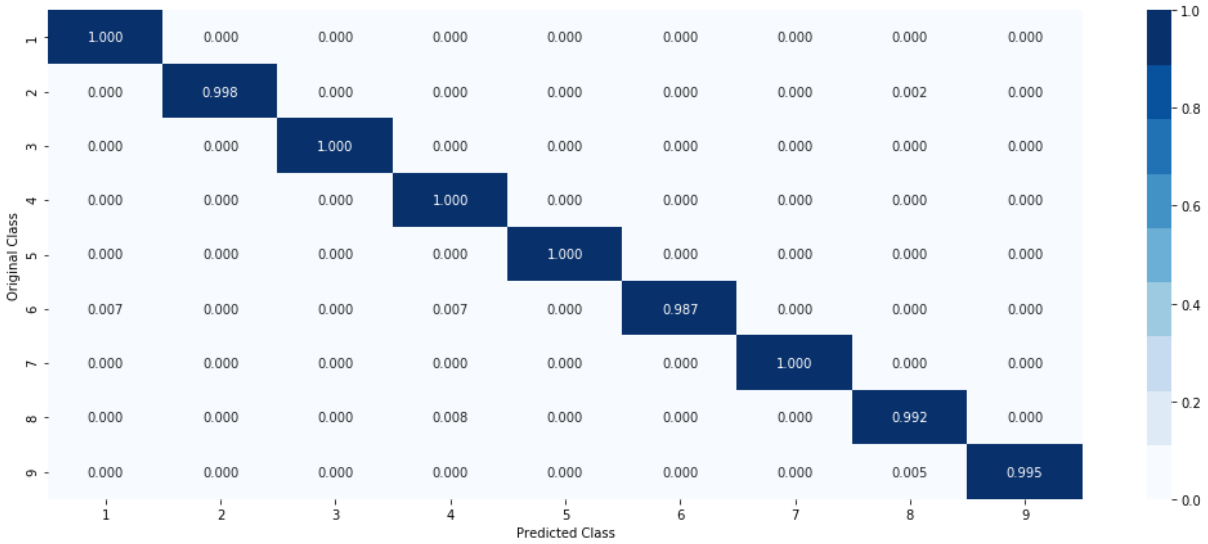
```
Log-Loss for Train-set is : 0.009665516643281273
Log-Loss for Test-set is : 0.017987329078114848
Train Accuracy is : 1.0
Test Accuracy is : 0.9981600735970562
---------------------------------------- CONFUSION-MATRIX ------------
--------------------------
```



---------- PRECISION-MATRIX ----------



---------- RECALL-MATRIX ----------



Conclusion:

- Advanced features helped a lot in reducing log-loss even further to 0.01.
- XGBOOST Model classifier performed the best among other models with log-loss of 0.0179.