In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

1.1 Reading Data

In [2]:

```
df = pd.read_csv(r'D:\ML Data\Hacker earth\Dataset\hm_train.csv')
df.head(5)
```

Out[2]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category
0	27673	24h	I went on a successful date with someone I fel...	1	affection
1	27674	24h	I was happy when my son got 90% marks in his e...	1	affection
2	27675	24h	I went to the gym this morning and did yoga.	1	exercise
3	27676	24h	We had a serious talk with some friends of our...	2	bonding
4	27677	24h	I went with grandchildren to butterfly display...	1	affection

In [3]:

```
print("Number of data points in train data", df.shape)
print('-'*50)
print("The attributes of data :", df.columns.values)
```

```
Number of data points in train data (60321, 5)
```

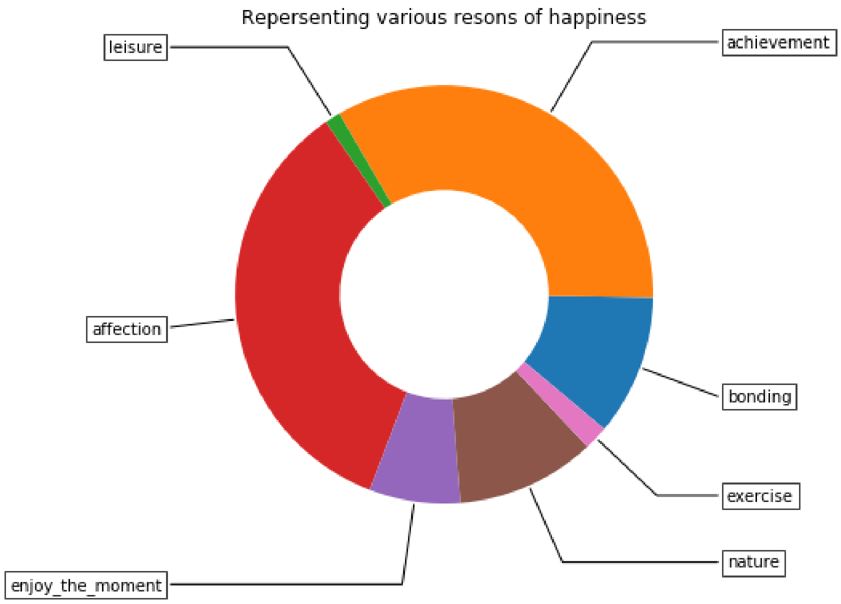
```
--------------------------------------------------
The attributes of data : ['hmid' 'reflection_period' 'cleaned_hm' 'num_sentence'
 'predicted_category']
```

1.2 Data Analysis

In [4]:

```
y_value_counts = df['predicted_category'].value_counts()
print("Number of peoples happy value is bonding ", y_value_counts['bonding'], ", (",
(y_value_counts['bonding']/(y_value_counts['bonding']+y_value_counts['achievement']+y_value_counts[
'affection']+y_value_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_counts['nature']+
y_value_counts['exercise']))*100,"%)")
print("Number of peoples happy value is achievement ", y_value_counts['achievement'], ", (",
(y_value_counts['achievement']/(y_value_counts['bonding']+y_value_counts['achievement']+y_value_cou
nts['affection']+y_value_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_counts['natur
e']+y_value_counts['exercise']))*100,"%)")
print("Number of peoples happy value is affection ", y_value_counts['affection'], ", (",
(y_value_counts['affection']/(y_value_counts['bonding']+y_value_counts['achievement']+y_value_count
s['affection']+y_value_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_counts['nature'
]+y_value_counts['exercise']))*100,"%)")
print("Number of peoples happy value is leisure ", y_value_counts['leisure'], ", (",
(y_value_counts['leisure']/(y_value_counts['bonding']+y_value_counts['achievement']+y_value_counts[
'affection']+y_value_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_counts['nature']+
y_value_counts['exercise']))*100,"%)")
print("Number of peoples happy value is enjoy the moment ", y_value_counts['enjoy_the_moment'], ",
(", (y_value_counts['enjoy_the_moment']/(y_value_counts['bonding']+y_value_counts['achievement']+y_
value_counts['affection']+y_value_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_coun
ts['nature']+y_value_counts['exercise']))*100,"%)")
print("Number of peoples happy value is nature ", y_value_counts['nature'], ", (", (y_value_counts[
'nature']/(y_value_counts['bonding']+y_value_counts['achievement']+y_value_counts['affection']+y_va
lue_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_counts['nature']+y_value_counts['e
xercise']))*100,"%)")
print("Number of peoples happy value is exercise ", y_value_counts['exercise'], ", (",
(y_value_counts['exercise']/(y_value_counts['bonding']+y_value_counts['achievement']+y_value_counts
['affection']+y_value_counts['leisure']+y_value_counts['enjoy_the_moment']+y_value_counts['nature']
+y_value_counts['exercise']))*100,"%)")

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
recipe = ['bonding', 'achievement', 'leisure','affection' , 'enjoy_the_moment', 'nature', 'exercise
']

data = [y_value_counts['bonding'], y_value_counts['achievement'], y_value_counts['exercise'],
y_value_counts['affection'], y_value_counts['leisure'], y_value_counts['enjoy_the_moment'],
y_value_counts['nature']]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)

bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                 horizontalalignment=horizontalalignment, **kw)

ax.set_title("Repersenting various resons of happiness")

plt.show()
```

```
Number of peoples happy value is bonding  6561 , ( 10.876809071467648 %)
Number of peoples happy value is achievement  20274 , ( 33.61018550753469 %)
Number of peoples happy value is affection  20880 , ( 34.61481076242105 %)
Number of peoples happy value is leisure  4242 , ( 7.032376784204505 %)
Number of peoples happy value is enjoy the moment  6508 , ( 10.78894580660135 %)
Number of peoples happy value is nature  1127 , ( 1.8683377264965766 %)
Number of peoples happy value is exercise  729 , ( 1.2085343412741831 %)
```



Observations: From above pie graph we conclude that affection appears most off the time in predicted values means lot of peoples happy because of affection. the order of appearance is:

affection > achievement > bonding > enjoy the moment > leisure > nature > exercise

In [5]:

```
#changing predicted_category to integer so that it fits in model easily
import operator
from tqdm import tqdm
import os

predict = df['predicted_category']
print(predict[6])
pre = predict.tolist()
#predict = np.asarray(pre)
print(type(predict))
count=0
y = ['achievement' , 'affection' , 'bonding' , 'enjoy_the_moment' , 'leisure' ,'nature' ,'exercise'
]
for i in tqdm(pre):
    if(operator.eq(i,y[0])):
        predict[count]=1
    elif(operator.eq(i,y[1])):
        predict[count]=2
    elif(operator.eq(i,y[2])):
        predict[count]=3
    elif(operator.eq(i,y[3])):
        predict[count]=4
    elif(operator.eq(i,y[4])):
        predict[count]=5
    elif(operator.eq(i,y[5])):
        predict[count]=6
    else:
        predict[count]=7
    count=count+1
```

```
achievement
<class 'pandas.core.series.Series'>
```

```
100%|██████████████████████████████████████████████████████████████████████████| 60321/60321 [31
:43<00:00, 31.69it/s]
```

In [6]:

```
pd.Series(predict)
df['predict']= predict
df.head(5)
```

Out[6]:

Out[6]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category	predict
0	27673	24h	I went on a successful date with someone I fel...	1	2	2
1	27674	24h	I was happy when my son got 90% marks in his e...	1	2	2
2	27675	24h	I went to the gym this morning and did yoga.	1	7	7
3	27676	24h	We had a serious talk with some friends of our...	2	3	3
4	27677	24h	I went with grandchildren to butterfly display...	1	2	2

In [7]:

```
df.head(5)
```

Out[7]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category	predict
0	27673	24h	I went on a successful date with someone I fel...	1	2	2
1	27674	24h	I was happy when my son got 90% marks in his e...	1	2	2
2	27675	24h	I went to the gym this morning and did yoga.	1	7	7
3	27676	24h	We had a serious talk with some friends of our...	2	3	3
4	27677	24h	I went with grandchildren to butterfly display...	1	2	2

1.3 Preprocessing of Cleaned_hm

In [8]:

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then retain that word from the data
            if not word in stop_words:
                string += word + " "

        df[column][index] = string
```

In [9]:

```
statement =df['cleaned_hm']
statement[10]
```

Out[9]:

```
'I came in 3rd place in my Call of Duty video game.'
```

In [10]:

```
#text processing stage.
start_time = time.clock()
for index, row in tqdm(df.iterrows()):
    if type(row['cleaned_hm']) is str:
        nlp_preprocessing(row['cleaned_hm'], index, 'cleaned_hm')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
60321it [33:23, 24.39it/s]
```

```
Time took for preprocessing the text : 2003.7545740268642 seconds
```

In [11]:

```
df[df.isnull().any(axis=1)] #checking if null value exists or not
```

Out[11]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category	predict

1.4. Test, Train and Cross Validation Split

1.4.1 Splitting data into train, test and cross validation (64:20:16)

In [12]:

```
y_true = df['predict'].values
# split the data into test and train by maintaining same distribution of output varaible 'y_true'
[stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(df, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output
varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
)
```

In [13]:

```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

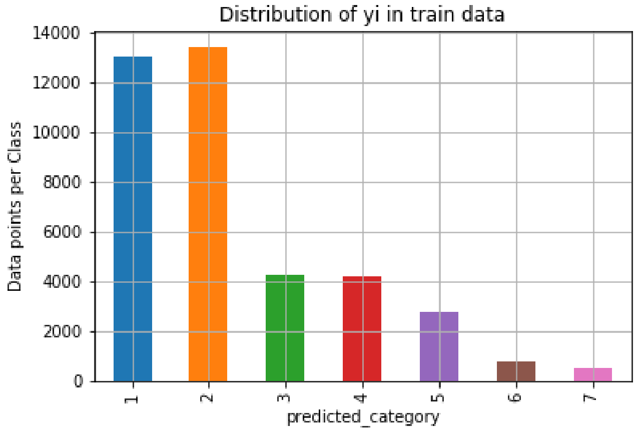
```
Number of data points in train data: 38604
Number of data points in test data: 12065
Number of data points in cross validation data: 9652
```

1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

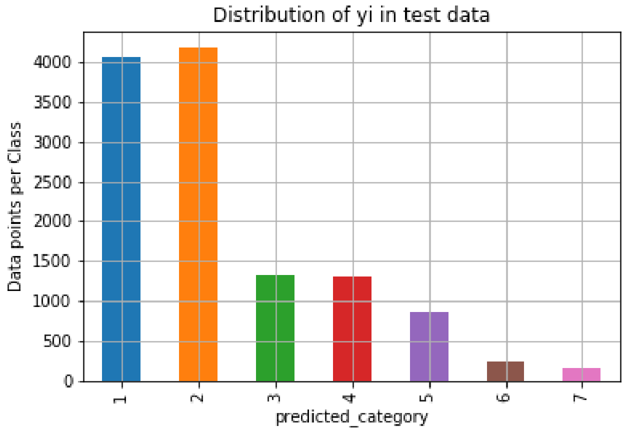
In [14]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['predicted_category'].value_counts().sortlevel()
test_class_distribution = test_df['predicted_category'].value_counts().sortlevel()
cv_class_distribution = cv_df['predicted_category'].value_counts().sortlevel()
y = ['achievement' , 'affection' , 'bonding' , 'enjoy the moment' , 'leisure' ,'nature' ,'exercise'
]
my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('predicted_category')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', y[i], ':',train_class_distribution.values[i], '(', np.r
ound((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
```

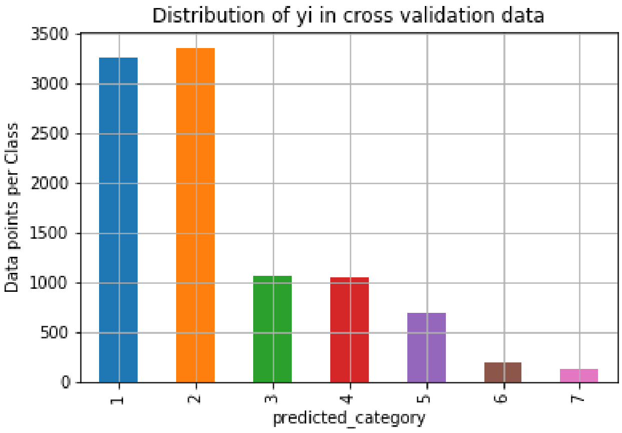
```
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('predicted_category')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', y[i], ':',test_class_distribution.values[i], '(', np.ro
und((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('predicted_category')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', y[i], ':',cv_class_distribution.values[i], '(', np.roun
d((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```



```
Number of data points in class affection : 13363 ( 34.616 %)
Number of data points in class achievement : 12975 ( 33.611 %)
Number of data points in class bonding : 4199 ( 10.877 %)
Number of data points in class enjoy the moment : 4165 ( 10.789 %)
Number of data points in class leisure : 2714 ( 7.03 %)
Number of data points in class nature : 722 ( 1.87 %)
Number of data points in class exercise : 466 ( 1.207 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class affection : 4176 ( 34.613 %)
Number of data points in class achievement : 4055 ( 33.61 %)
Number of data points in class achievement : 4055 ( 33.61 %)
Number of data points in class bonding : 1312 ( 10.874 %)
Number of data points in class enjoy the moment : 1302 ( 10.792 %)
Number of data points in class leisure : 849 ( 7.037 %)
Number of data points in class nature : 225 ( 1.865 %)
Number of data points in class exercise : 146 ( 1.21 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class affection : 3341 ( 34.615 %)
Number of data points in class achievement : 3244 ( 33.61 %)
Number of data points in class bonding : 1050 ( 10.879 %)
Number of data points in class enjoy the moment : 1041 ( 10.785 %)
Number of data points in class leisure : 679 ( 7.035 %)
Number of data points in class nature : 180 ( 1.865 %)
Number of data points in class exercise : 117 ( 1.212 %)
```

1.5 Prediction using a 'Random' Model

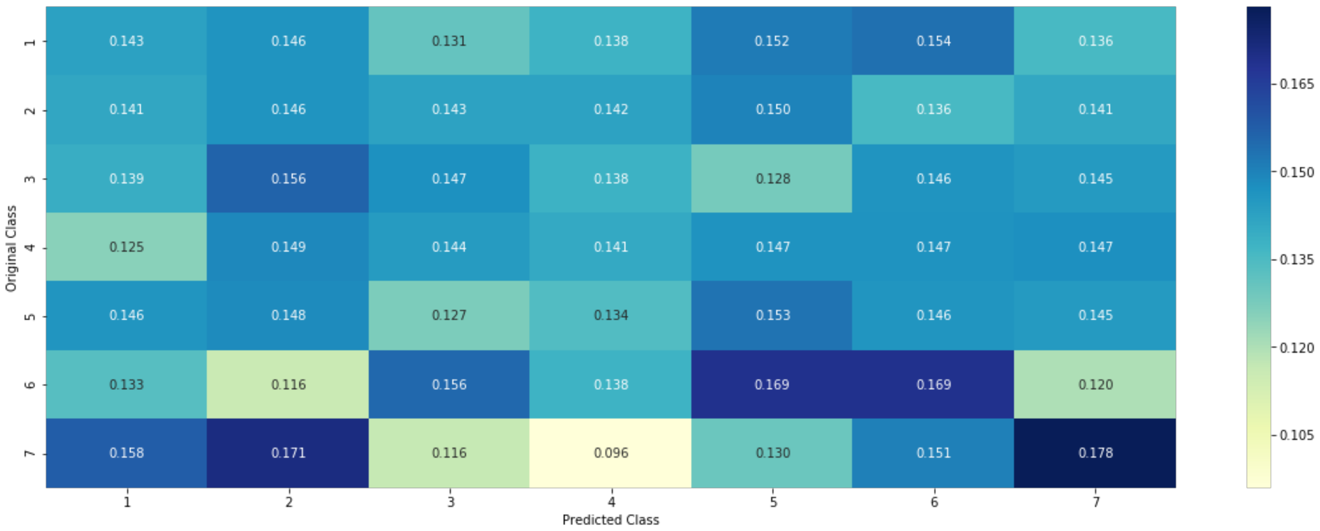
In [15]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))

    F1 = 2*((A*B)/(A+B))

    labels = [1,2,3,4,5,6,7]

   # print("-"*20, "Confusion matrix", "-"*20)
   # plt.figure(figsize=(20,7))
   # sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
   # plt.xlabel('Predicted Class')
   # plt.ylabel('Original Class')
   # plt.show()
    # representing A in heatmap format
    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "F1-Score matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(F1, annot=True, cmap="YlGnBu",fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

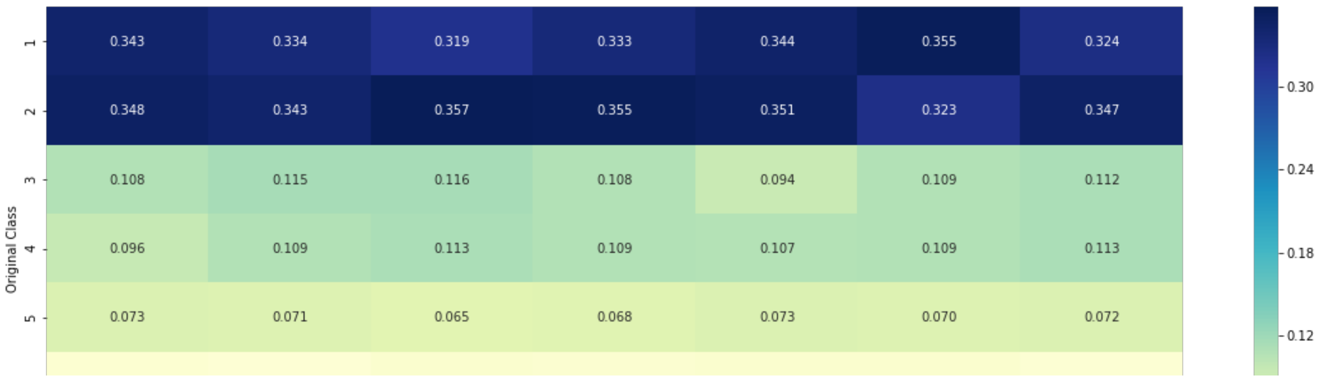
In [16]:

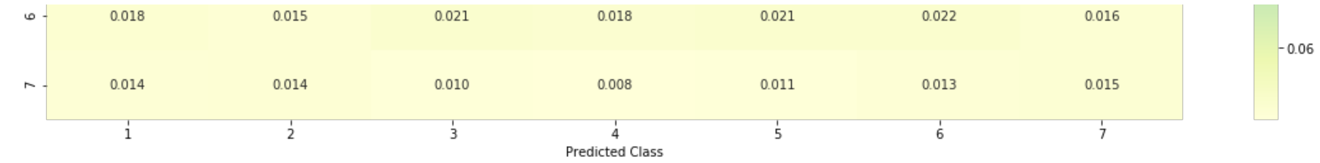
```
from sklearn.metrics import precision_recall_fscore_support as score
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,7))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,7)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
    y_cv=y_cv.astype('int')
    type(cv_predicted_y)
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-
15))
print(type(cv_predicted_y))

# Test-Set error.
#we create a output array that has exactly same size as the test data
test_predicted_y = np.zeros((test_data_len,7))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,7)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
    y_test=y_test.astype('int')
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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 2.2336741349138083
<class 'numpy.ndarray'>
Log loss on Test Data using Random Model 2.217171308271734
-------------------- Precision matrix (Columm Sum=1) --------------------
```

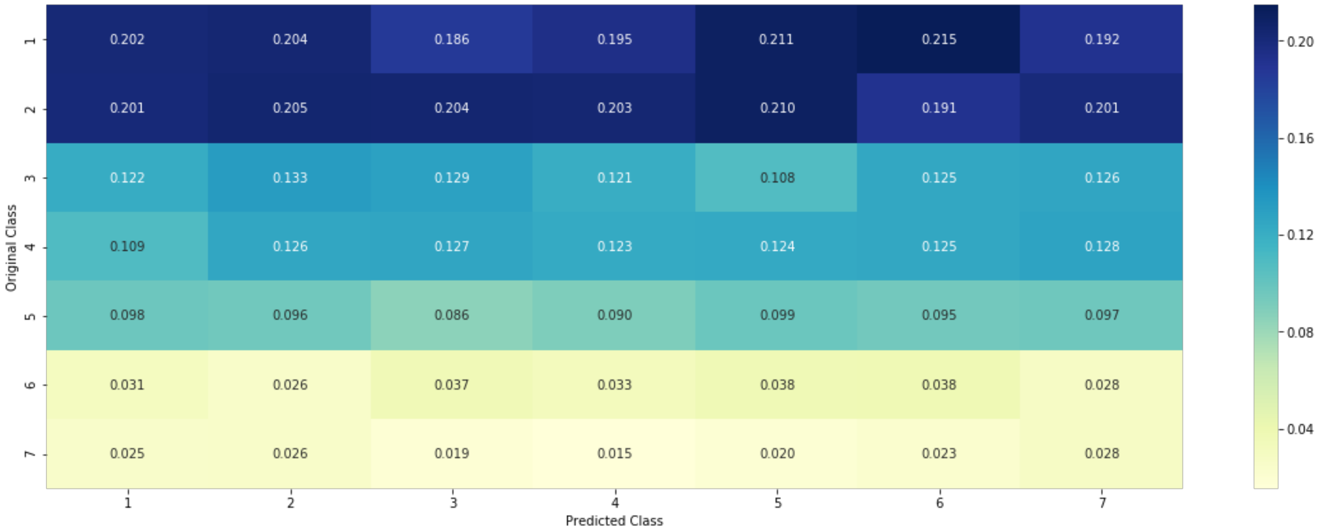


```
-------------------- Recall matrix (Row sum=1) --------------------
```





```
-------------------- F1-Score matrix (Row sum=1) --------------------
```



Univariate Analysis on cleaned_hm

In [17]:

```
def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['cleaned_hm'].split():
            dictionary[word] +=1
    return dictionary
```

In [18]:

```
# building a tf-idf Vectorizer with all the words that occured minimum 3 times in train data
cleaned_vectorizer = TfidfVectorizer()
train_vectorizer = cleaned_vectorizer.fit_transform(train_df['cleaned_hm'])

train_text_features= cleaned_vectorizer.get_feature_names()

print("Total number of unique words in train data :", len(train_text_features))
```

```
Total number of unique words in train data : 16969
```

In [19]:

```
dict_list = []
# dict_list =[] contains 7 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['predict']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

In [21]:

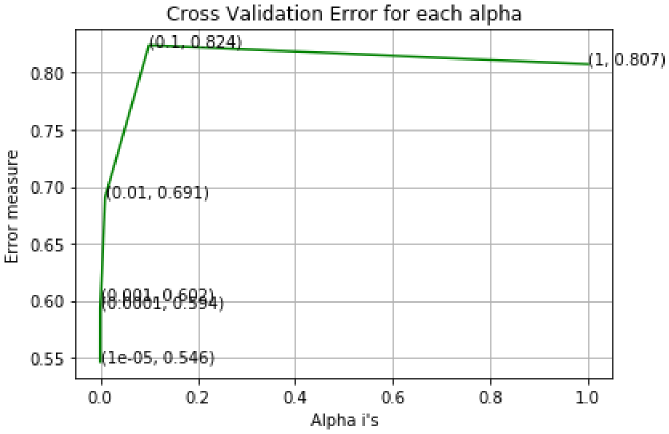
```
# don't forget to normalize every feature
train_vectorizer = normalize(train_vectorizer, axis=0)
print(train_vectorizer.shape)
# we use the same vectorizer that was trained on train data
test_vectorizer = cleaned_vectorizer.transform(test_df['cleaned_hm'])
# don't forget to normalize every feature
test_vectorizer = normalize(test_vectorizer, axis=0)
print(test_vectorizer.shape)

# we use the same vectorizer that was trained on train data
cv_vectorizer = cleaned_vectorizer.transform(cv_df['cleaned_hm'])
# don't forget to normalize every feature
cv_vectorizer = normalize(cv_vectorizer, axis=0)
print(cv_vectorizer.shape)
```

```
(38604, 16969)
(12065, 16969)
(9652, 16969)
```

In [22]:

```
# Train a Logistic regression+Calibration model
alpha = [10 ** x for x in range(-5, 1)]

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    y_train=y_train.astype('int')
    clf.fit(train_vectorizer, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_vectorizer, y_train)
    predict_y = sig_clf.predict_proba(cv_vectorizer)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))

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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
y_train=y_train.astype('int')
clf.fit(train_vectorizer, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_vectorizer, y_train)

predict_y = sig_clf.predict_proba(train_vectorizer)
y_train=y_train.astype('int')
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_vectorizer)
y_cv=y_cv.astype('int')
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_vectorizer)
y_test=y_test.astype('int')
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 0.546133629958419
For values of alpha =  0.0001 The log loss is: 0.5937566930150481
For values of alpha =  0.001 The log loss is: 0.6015261201069694
For values of alpha =  0.01 The log loss is: 0.6908385144215637
For values of alpha =  0.1 The log loss is: 0.8241230038992128
For values of alpha =  1 The log loss is: 0.8074572487456985
```



```
For values of best alpha =  1e-05 The train log loss is: 0.30671817608289315
For values of best alpha =  1e-05 The cross validation log loss is: 0.546133629958419
For values of best alpha =  1e-05 The test log loss is: 0.5412728224811045
```

Q. Is the cleaned_hm feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it seems like!

In [23]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['cleaned_hm'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [24]:

```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
99.337 % of word of test data appeared in train data
99.733 % of word of Cross Validation appeared in train data
```

2. K Nearest Neighbour Classification

In [25]:

```
#Data preparation for ML models.
#Misc. functionns for ML models
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y - test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [26]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [27]:

```
train_y = np.array(list(train_df['predict']))

test_y = np.array(list(test_df['predict']))

cv_y = np.array(list(cv_df['predict']))
```

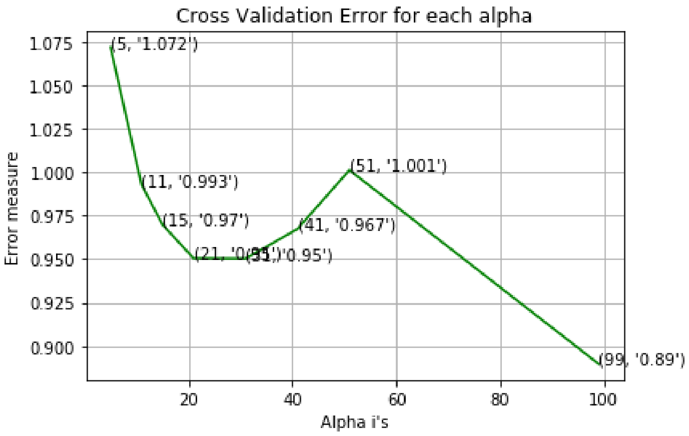
2.1.1 Hyper parameter tuning

In [28]:

```
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_vectorizer, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_vectorizer, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_vectorizer)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (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()


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_vectorizer, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_vectorizer, train_y)

predict_y = sig_clf.predict_proba(train_vectorizer)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_vectorizer)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_vectorizer)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0717606531256596
for alpha = 11
Log Loss : 0.9925631471670441
for alpha = 15
Log Loss : 0.9697841142515448
for alpha = 21
Log Loss : 0.9504322528751623
```

```
for alpha = 31
Log Loss : 0.9502736162301463
for alpha = 41
Log Loss : 0.9673089844657241
for alpha = 51
Log Loss : 1.0011397380487115
for alpha = 99
Log Loss : 0.8897196771191992
```



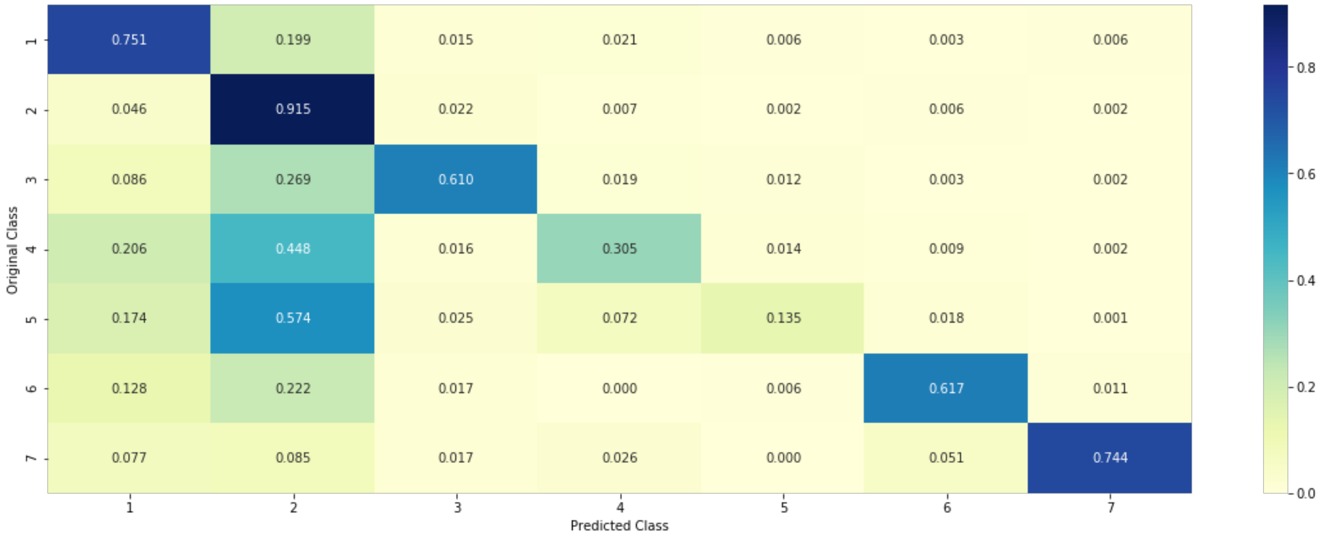
```
For values of best alpha =  99 The train log loss is: 0.9626072839880712
For values of best alpha =  99 The cross validation log loss is: 0.8897196771191992
For values of best alpha =  99 The test log loss is: 0.8813311268165666
```

2.1.2 Testing the model with best hyper paramters

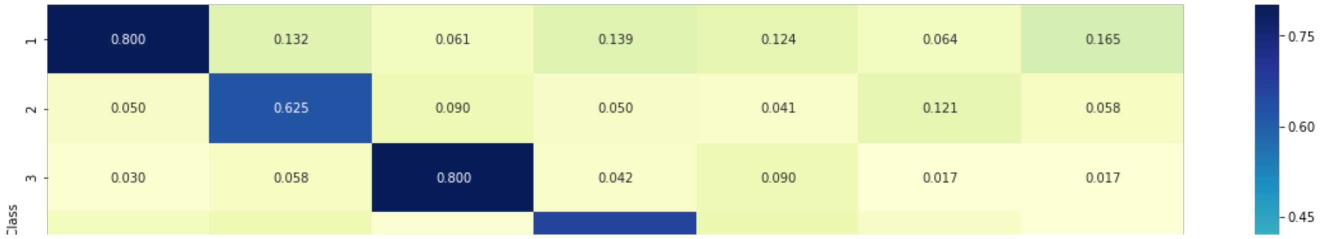
In [29]:

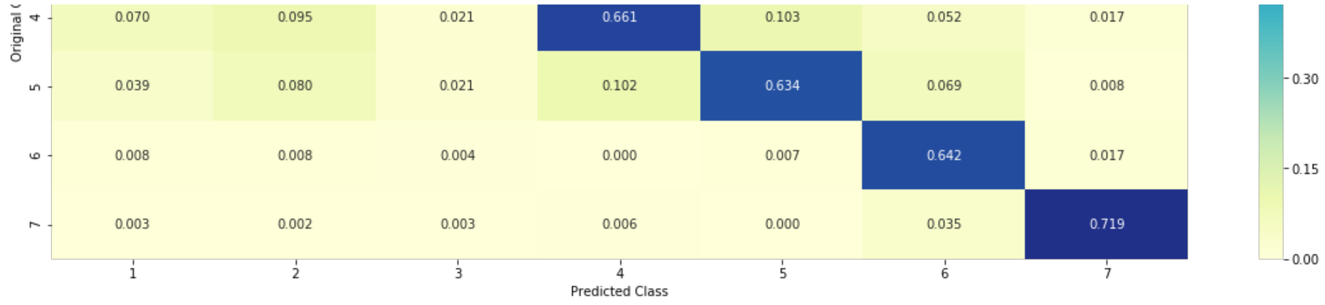
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_vectorizer, train_y, cv_vectorizer, cv_y, clf)
```

```
Log loss : 0.8897196771191992
Number of mis-classified points : 0.3015955242436801
-------------------- Precision matrix (Columm Sum=1) --------------------
```

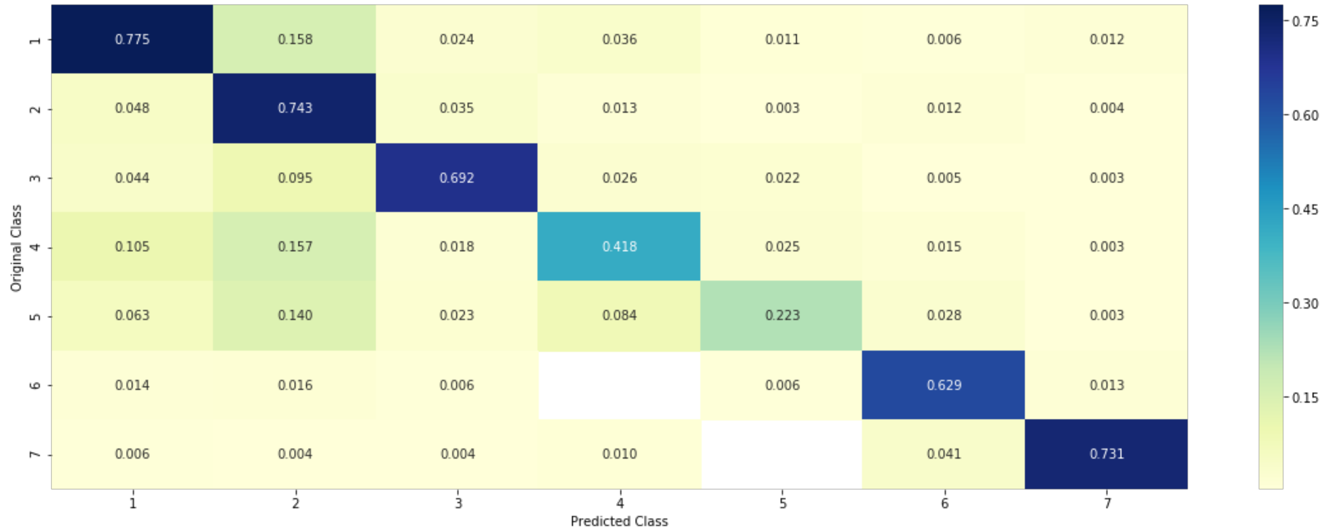


```
-------------------- Recall matrix (Row sum=1) --------------------
```





```
-------------------- F1-Score matrix (Row sum=1) --------------------
```



2.1.2 predicting the values for test_hm.csv

In [30]:

```
df_test_file = pd.read_csv(r'D:\ML Data\Hacker earth\Dataset\hm_test.csv')
print(df_test_file.head(2))


print(df_test_file.columns.values)
df_test_file.shape
```

```
    hmid reflection_period                                         cleaned_hm  \
0  88305                3m    I spent the weekend in Chicago with my friends.
1  88306                3m  We moved back into our house after a remodel. ...

   num_sentence
0             1
1             2
['hmid' 'reflection_period' 'cleaned_hm' 'num_sentence']
```

Out[30]:

```
(40213, 4)
```

In [31]:

```
test_vectorizer = cleaned_vectorizer.transform(df_test_file['cleaned_hm'])
print(test_vectorizer.shape)
```

```
(40213, 16969)
```

In [32]:

```
predict = sig_clf.predict(test_vectorizer)
```

In [33]:

```
print(df_test_file.shape)
predict.shape
```

```
(40213, 4)
```

Out[33]:

```
(40213,)
```

In [34]:

```
#import sklearn.preprocessing
print(predict)
#inverse = cleaned_vectorizer.inverse_transform(predict_y)
df_test_file['predicted_category'] = predict
```

```
[2 2 2 ... 2 2 7]
```

In [35]:

```
print(df_test_file.shape)
df_test_file.head(5)
```

```
(40213, 5)
```

Out[35]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category
0	88305	3m	I spent the weekend in Chicago with my friends.	1	2
1	88306	3m	We moved back into our house after a remodel. ...	2	2
2	88307	3m	My fiance proposed to me in front of my family...	1	2
3	88308	3m	I ate lobster at a fancy restaurant with some ...	1	4
4	88309	3m	I went out to a nice restaurant on a date with...	5	2

In [36]:

```
#converting predicted_category to string again
import operator
from tqdm import tqdm
import os
predict = df_test_file['predicted_category']
print(predict[6])
pre = predict.tolist()
#predict = np.asarray(pre)
print(type(predict))
count=0
y = ['achievement' , 'affection' , 'bonding' , 'enjoy_the_moment' , 'leisure' ,'nature' ,'exercise'
]
for i in tqdm(pre):
    if(i==1):
        predict[count]='achievement'
    elif(i==2):
        predict[count]='affection'
    elif(i==3):
        predict[count]='bonding'
    elif(i==4):
        predict[count]='enjoy_the_moment'
    elif(i==5):
        predict[count]='leisure'
    elif(i==6):
        predict[count]='nature'
    else:
        predict[count]='exercise'
    count=count+1
```

```
1
```

```
<class 'pandas.core.series.Series'>
```

```
100%|████████████████████████████████████████████████████████████| 40213/40213 [28
:41<00:00, 23.36it/s]
```

In [37]:

```
df_test_file['predicted_cat'] = predict
```

In [38]:

```
df_test_file.head(5)
```

Out[38]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category	predicted_cat
0	88305	3m	I spent the weekend in Chicago with my friends.	1	affection	affection
1	88306	3m	We moved back into our house after a remodel. ...	2	affection	affection
2	88307	3m	My fiance proposed to me in front of my family...	1	affection	affection
3	88308	3m	I ate lobster at a fancy restaurant with some ...	1	enjoy_the_moment	enjoy_the_moment
4	88309	3m	I went out to a nice restaurant on a date with...	5	affection	affection

In [39]:

```
df_test_file.drop(['predicted_cat'], 1, inplace=True)
```

In [40]:

```
df_test_file.head(5)
```

Out[40]:

	hmid	reflection_period	cleaned_hm	num_sentence	predicted_category
0	88305	3m	I spent the weekend in Chicago with my friends.	1	affection
1	88306	3m	We moved back into our house after a remodel. ...	2	affection
2	88307	3m	My fiance proposed to me in front of my family...	1	affection
3	88308	3m	I ate lobster at a fancy restaurant with some ...	1	enjoy_the_moment
4	88309	3m	I went out to a nice restaurant on a date with...	5	affection

In [41]:

```
df = pd.DataFrame(data={"hmid": df_test_file['hmid'], "predicted_category":
df_test_file['predicted_category']})
df.to_csv('D:\ML Data\Hacker earth\Dataset\subKnn.csv', sep=',',index=False)
sub = pd.read_csv(r'D:\ML Data\Hacker earth\Dataset\subKnn.csv')
sub.head(5)
```

Out[41]:

	hmid	predicted_category
0	88305	affection
1	88306	affection

	hmid	predicted_category
2	88307	affection
3	88308	enjoy_the_moment
4	88309	affection