Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

ID,Gene,Variation,Class
0,FAM58A,Truncating Mutations,1
1,CBL,W802*,2
2,CBL,Q249E,2
...

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

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 StratifiedKFold
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
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

```
Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

Out[2]:

ID	Gene	Variation	Class

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

training/training_variants is a comma separated file containing the description of the genetic mutations used for training.
Fields are

- **ID :** the id of the row used to link the mutation to the clinical evidence
- **Gene :** the gene where this genetic mutation is located
- **Variation :** the aminoacid change for this mutations
- **Class :** 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

In [3]:

```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skip
rows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

```
Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']
```

Out[3]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3. Preprocessing of text

In [7]:

```
# 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:
```

```
            if not word in stop_words:
                string += word + " "

        data_text[column][index] = string
```

In [8]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 143.34249110000002 seconds
```

In [4]:

```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[4]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

In [5]:

```
result[result.isnull().any(axis=1)]
```

Out[5]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

In [6]:

```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [7]:

```
result[result['ID']==1109]
```

Out[7]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

3.1.4. Test, Train and Cross Validation Split

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

In [8]:

```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# 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(result, 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
)
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [9]:

```
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])
```

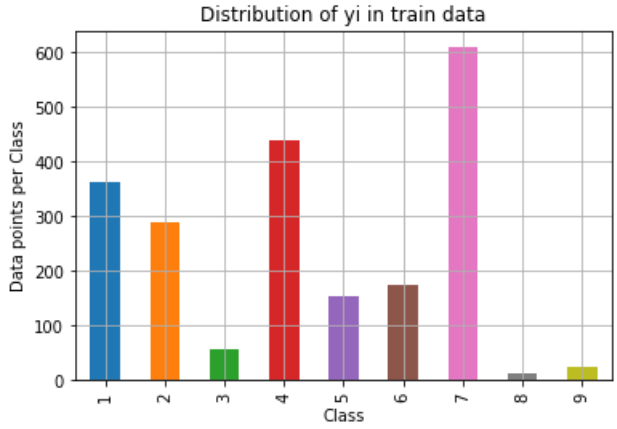
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

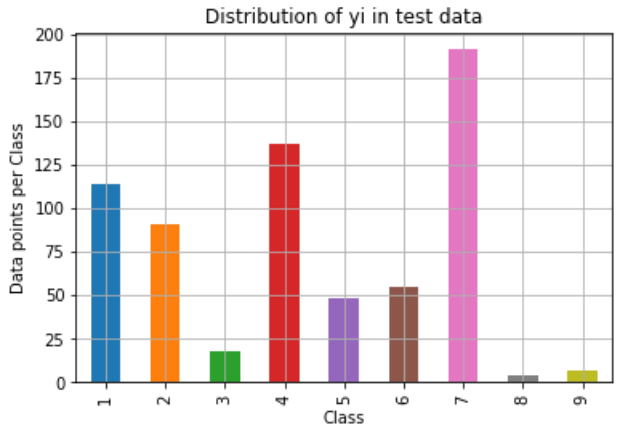
In [0]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(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', i+1, ':',train_class_distribution.values[i], '(', np.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


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

```
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(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', i+1, ':',test_class_distribution.values[i], '(', np.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(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', i+1, ':',cv_class_distribution.values[i], '(', np.round
((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```

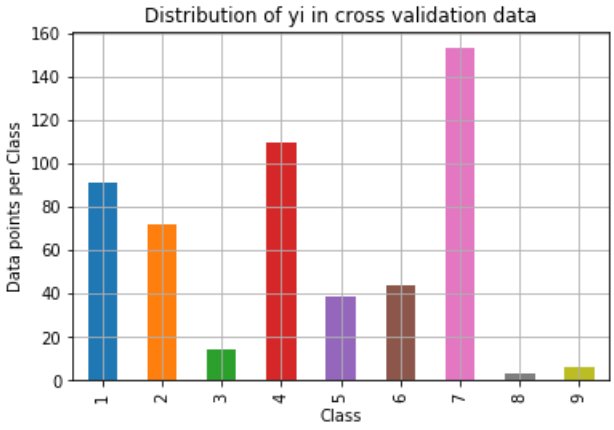



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
```

```
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
------------------------------------------------------------------------------------------
```




```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [10]:

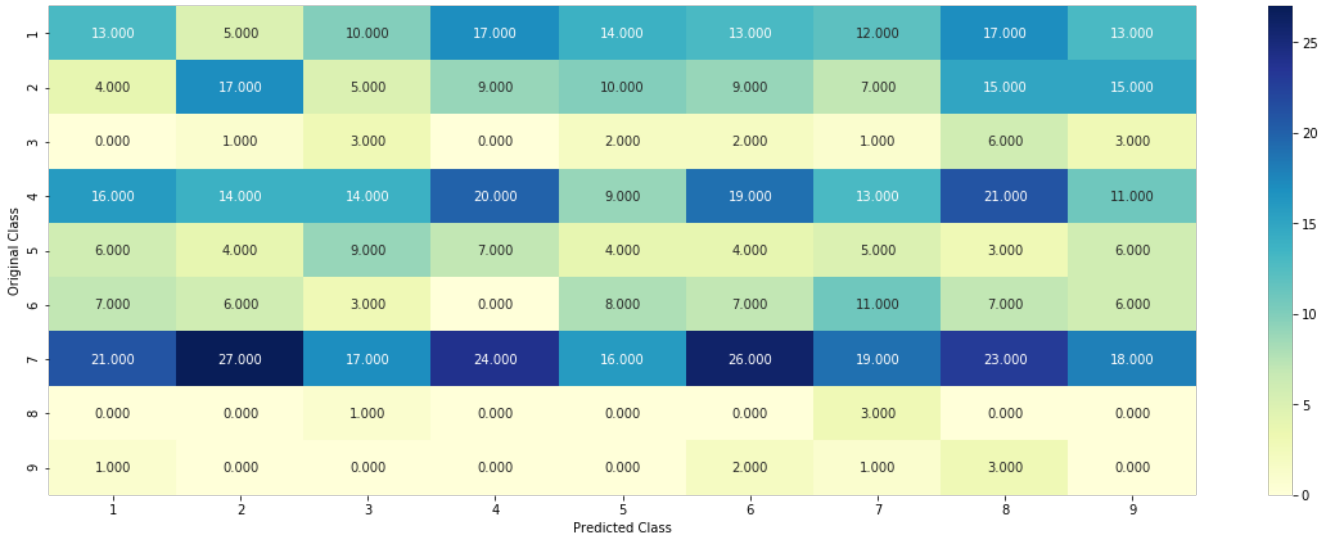
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    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()

    print("-"*20, "Precision matrix (Columm 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()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row 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()
```

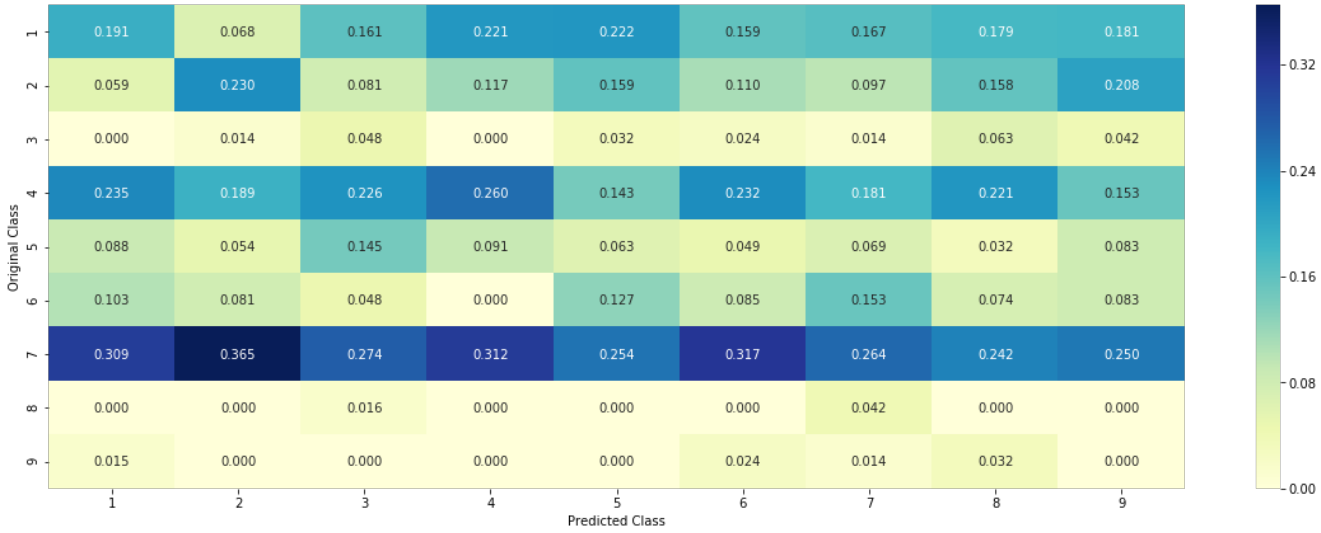
In [0]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
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,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

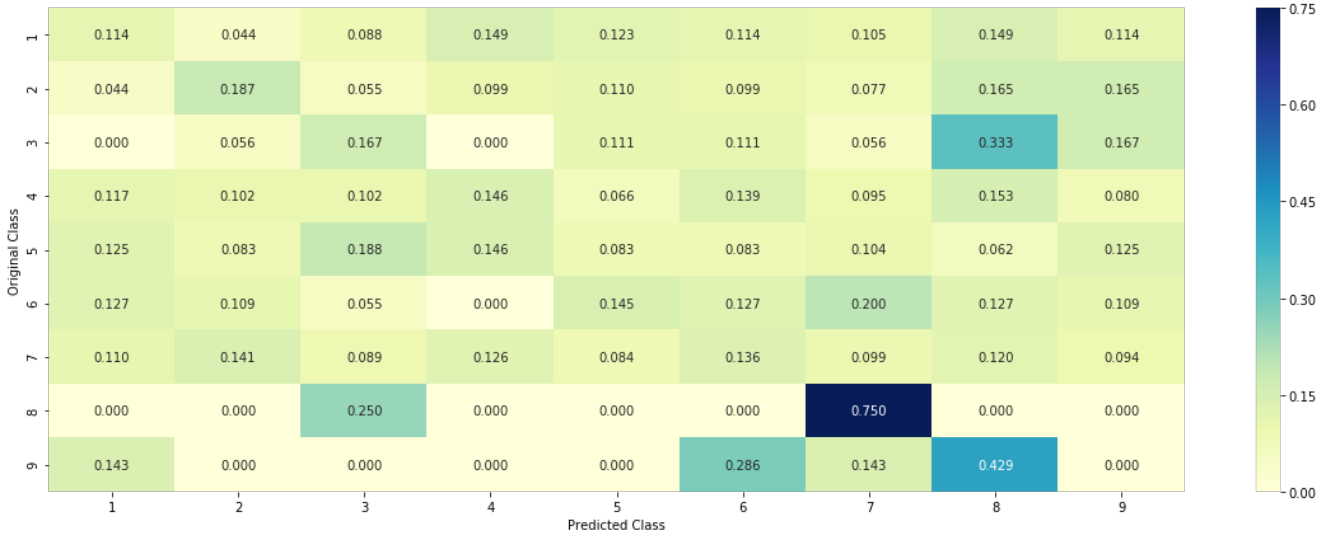
```
Log loss on Cross Validation Data using Random Model 2.536598785706848
Log loss on Test Data using Random Model 2.501572555849742
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



3.3 Univariate Analysis

In [31]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / nu
mber of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
```

```
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #          ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
```

```
    #       'BRAF': [0.06666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.06666666666666666, 0.06666666666666666],
    #       ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the da
ta
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- (numerator + 10*alpha) / (denominator + 90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

In [0]:

```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 229
BRCA1     164
TP53       90
EGFR       88
PTEN       82
BRCA2      78
KIT        67
BRAF       51
ALK        48
ERBB2      46
PIK3CA     37
Name: Gene, dtype: int64
```

In [0]:

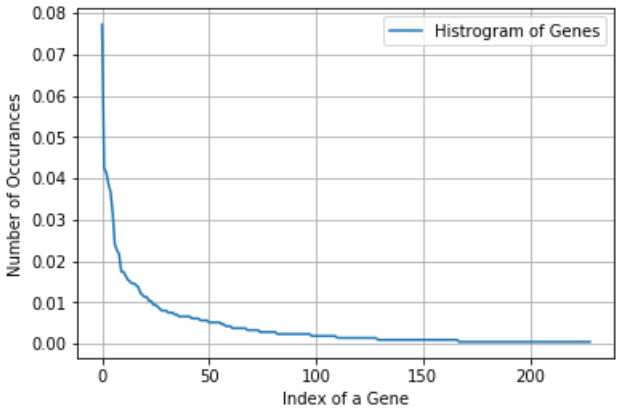
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
d they are distibuted as follows",)
```

```
Ans: There are 229 different categories of genes in the train data, and they are distibuted as fol
lows
```

In [0]:

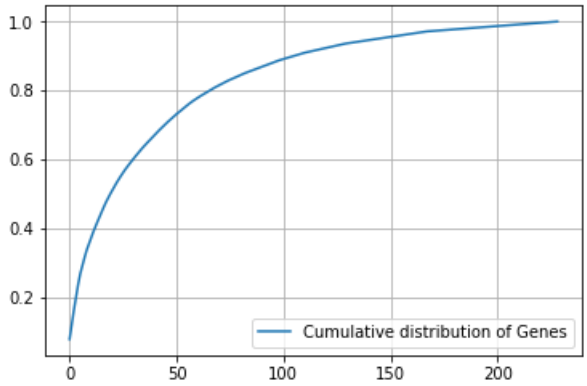
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
```

```
plt.legend()
plt.grid()
plt.show()
```



In [0]:

```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

In [32]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [0]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The sha
```

```
pe of gene feature:", train_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape of g
ene feature: (2124, 9)
```

In [20]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [0]:

```
train_df['Gene'].head()
```

Out[0]:

```
2988       KIT
1718    KNSTRN
2076      TET2
751      ERBB2
462       TP53
Name: Gene, dtype: object
```

In [0]:

```
gene_vectorizer.get_feature_names()
```

Out[0]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
```

```
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'gata3',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
```

```
'kras',
'lats1',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm4',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk3',
'nup93',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhb',
```

```
 'sdhc',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [0]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The sha
pe of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 229)
```

Q4. How good is this gene feature in predicting y_i?

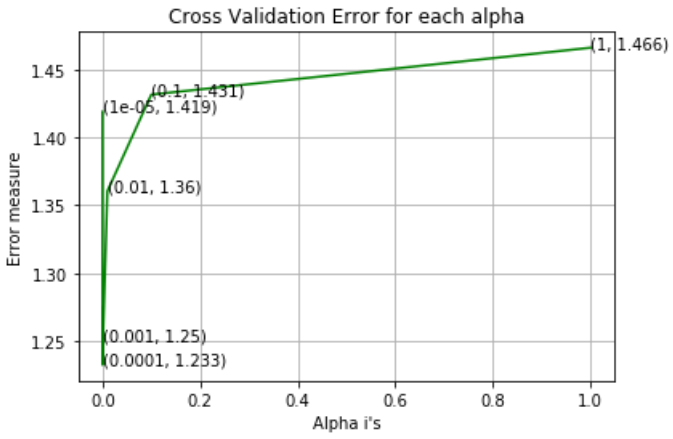
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [0]:

```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link:
#-------------------------------


cv_log_error_array=[]
for i in alpha:
```

```
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    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)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
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_gene_feature_onehotCoding)
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_gene_feature_onehotCoding)
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: 1.418841767162939
For values of alpha =  0.0001 The log loss is: 1.2325868001617826
For values of alpha =  0.001 The log loss is: 1.2503129272158073
For values of alpha =  0.01 The log loss is: 1.360379976757511
For values of alpha =  0.1 The log loss is: 1.4314392521126913
For values of alpha =  1 The log loss is: 1.4659143358159061
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0425604300119806
For values of best alpha =  0.0001 The cross validation log loss is: 1.2325868001617826
For values of best alpha =  0.0001 The test log loss is: 1.200905436534172
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

In [0]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0
], " genes in train dataset?")
```

```
test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  229  genes in train dataset?
Ans
1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 514 out of  532 : 96.61654135338345
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [0]:

```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1924
Truncating_Mutations         59
Deletion                     49
Amplification                47
Fusions                      23
E17K                          3
Overexpression                3
Q22K                          2
Promoter_Hypermethylation     2
G13D                          2
T73I                          2
Name: Variation, dtype: int64
```

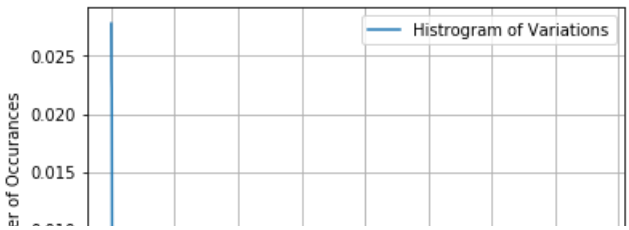
In [0]:

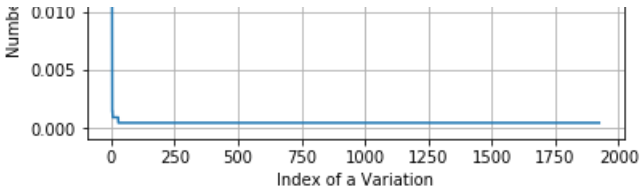
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1924 different categories of variations in the train data, and they are distibuted
as follows
```

In [0]:

```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```

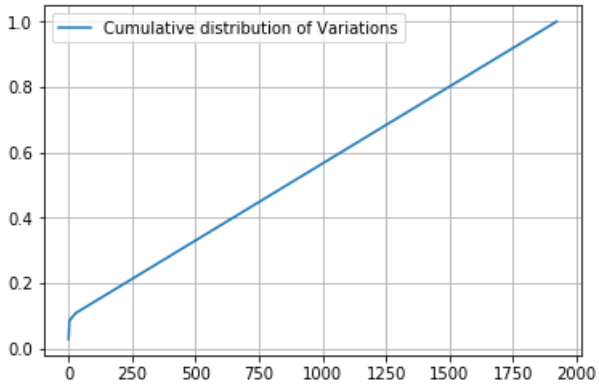




In [0]:

```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02777778 0.05084746 0.07297552 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

We will be using both these methods to featurize the Variation Feature

In [33]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [0]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

```
train_variation_feature_responseCoding is a converted feature using the response coding method. Th
e shape of Variation feature: (2124, 9)
```

In [21]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
```

```
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [0]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
od. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1960)
```

Q10. How good is this Variation feature in predicting y_i?

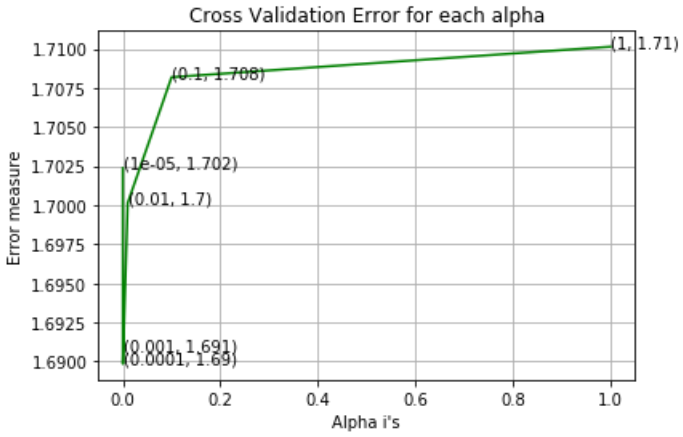
Let's build a model just like the earlier!

In [0]:

```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link:
#-------------------------------


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

    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)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
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_variation_feature_onehotCoding)
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_variation_feature_onehotCoding)
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: 1.7023621747765858
For values of alpha =  0.0001 The log loss is: 1.689842322110077
For values of alpha =  0.001 The log loss is: 1.6907130717518253
For values of alpha =  0.01 The log loss is: 1.7001345396142153
For values of alpha =  0.1 The log loss is: 1.7081826775989355
For values of alpha =  1 The log loss is: 1.710131025593559
```




```
For values of best alpha =  0.0001 The train log loss is: 0.8255455900343496
For values of best alpha =  0.0001 The cross validation log loss is: 1.689842322110077
For values of best alpha =  0.0001 The test log loss is: 1.7362385768757977
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Not sure! But lets be very sure using the below analysis.

In [0]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
st and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q12. How many data points are covered by total  1924  genes in test and cross validation data
sets?
Ans
1. In test data 64 out of 665 : 9.624060150375941
2. In cross validation data 56 out of  532 : 10.526315789473683
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

In [11]:

```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
```

```
# increment its count whenever we see that word

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

In [12]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [23]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

```
Total number of unique words in train data : 57471
```

In [26]:

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==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 [27]:

```
#response coding of text features
```

```
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [0]:

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding =
(test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [28]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

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

In [0]:

```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [0]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 6168, 4: 3775, 5: 3025, 6: 2977, 9: 2140, 8: 1985, 7: 1945, 10: 1323, 12: 1133, 11: 110
7, 13: 1068, 15: 1059, 14: 873, 16: 851, 18: 747, 17: 575, 24: 574, 20: 552, 21: 509, 19: 495, 22:
466, 25: 415, 37: 413, 27: 396, 28: 394, 23: 380, 30: 378, 26: 319, 45: 312, 29: 284, 34: 282, 35:
275, 31: 274, 32: 259, 36: 253, 33: 240, 44: 231, 40: 225, 39: 220, 48: 206, 42: 204, 38: 204, 56:
177, 47: 173, 43: 173, 46: 171, 41: 168, 51: 161, 50: 159, 60: 143, 53: 140, 52: 136, 49: 133, 57:
132, 55: 131, 54: 128, 70: 120, 67: 116, 58: 113, 74: 111, 66: 108, 62: 105, 59: 104, 88: 102, 64:
101, 61: 100, 63: 99, 65: 98, 78: 97, 75: 96, 72: 96, 68: 94, 80: 93, 69: 92, 79: 85, 73: 81, 71:
79, 91: 77, 82: 77, 86: 76, 83: 76, 90: 75, 77: 73, 84: 72, 93: 69, 81: 69, 76: 68, 95: 65, 96: 6
4, 87: 63, 92: 59, 98: 58, 85: 58, 120: 57, 99: 56, 115: 55, 100: 55, 108: 54, 102: 54, 107: 53,
89: 53, 132: 52, 116: 52, 109: 52, 105: 52, 101: 51, 111: 50, 97: 50, 94: 50, 140: 48, 110: 48, 10
4: 48, 103: 48, 117: 45, 113: 45, 144: 44, 121: 43, 141: 42, 126: 42, 114: 42, 106: 42, 119: 41, 1
34: 40, 136: 39, 130: 39, 135: 38, 131: 38, 128: 38, 112: 38, 148: 36, 129: 35, 125: 35, 124: 35,
122: 35, 161: 34, 150: 34, 147: 34, 145: 34, 139: 34, 127: 34, 149: 33, 137: 33, 118: 33, 185: 32,
165: 32, 157: 32, 133: 32, 166: 30, 164: 30, 142: 30, 155: 29, 151: 29, 219: 28, 170: 28, 167: 28,
143: 28, 169: 27, 195: 26, 181: 26, 177: 26, 172: 26, 123: 26, 215: 25, 184: 25, 160: 25, 154: 25,
152: 25, 138: 25, 253: 24, 189: 24, 183: 24, 175: 24, 162: 24, 156: 24, 225: 23, 212: 23, 191: 23,
186: 23, 180: 23, 176: 23, 287: 22, 216: 22, 193: 22, 171: 22, 158: 22, 276: 21, 254: 21, 228: 21,
182: 21, 292: 20, 281: 20, 234: 20, 231: 20, 222: 20, 220: 20, 194: 20, 187: 20, 173: 20, 153: 20,
242: 19, 240: 19, 230: 19, 223: 19, 206: 19, 204: 19, 197: 19, 179: 19, 163: 19, 146: 19, 288: 18,
261: 18, 246: 18, 226: 18, 224: 18, 210: 18, 201: 18, 196: 18, 174: 18, 168: 18, 283: 17, 255: 17,
236: 17, 221: 17, 209: 17, 208: 17, 202: 17, 192: 17, 159: 17, 320: 16, 270: 16, 251: 16, 247: 16,
238: 16, 237: 16, 218: 16, 217: 16, 214: 16, 207: 16, 203: 16, 199: 16, 318: 15, 306: 15, 272: 15,
259: 15, 252: 15, 245: 15, 229: 15, 188: 15, 178: 15, 358: 14, 328: 14, 313: 14, 300: 14, 294: 14,
279: 14, 278: 14, 274: 14, 265: 14, 239: 14, 213: 14, 365: 13, 339: 13, 333: 13, 315: 13, 307: 13,
302: 13, 295: 13, 286: 13, 266: 13, 263: 13, 257: 13, 250: 13, 232: 13, 227: 13, 211: 13, 205: 13,
190: 13, 474: 12, 338: 12, 317: 12, 310: 12, 285: 12, 277: 12, 269: 12, 268: 12, 258: 12, 244: 12,
241: 12, 387: 11, 373: 11, 350: 11, 331: 11, 326: 11, 314: 11, 311: 11, 309: 11, 282: 11, 280: 11,
273: 11, 262: 11, 249: 11, 243: 11, 200: 11, 198: 11, 466: 10, 452: 10, 435: 10, 417: 10, 403: 10,
393: 10, 390: 10, 371: 10, 364: 10, 363: 10, 359: 10, 357: 10, 354: 10, 347: 10, 346: 10, 345: 10,
324: 10, 319: 10, 303: 10, 299: 10, 298: 10, 296: 10, 275: 10, 235: 10, 233: 10, 822: 9, 541: 9, 5
31: 9, 467: 9, 462: 9, 449: 9, 444: 9, 442: 9, 436: 9, 429: 9, 419: 9, 399: 9, 397: 9, 381: 9,
370: 9, 369: 9, 367: 9, 352: 9, 348: 9, 344: 9, 343: 9, 340: 9, 334: 9, 325: 9, 322: 9, 271: 9
```

370: 9, 369: 9, 367: 9, 352: 9, 348: 9, 344: 9, 343: 9, 340: 9, 334: 9, 325: 9, 322: 9, 271: 9,
845: 8, 720: 8, 646: 8, 570: 8, 566: 8, 477: 8, 447: 8, 421: 8, 416: 8, 414: 8, 407: 8, 405: 8,
404: 8, 374: 8, 366: 8, 362: 8, 360: 8, 355: 8, 337: 8, 336: 8, 323: 8, 308: 8, 305: 8, 297: 8,
293: 8, 260: 8, 256: 8, 248: 8, 679: 7, 653: 7, 550: 7, 547: 7, 537: 7, 520: 7, 515: 7, 513: 7,
509: 7, 504: 7, 469: 7, 460: 7, 458: 7, 446: 7, 445: 7, 443: 7, 441: 7, 440: 7, 425: 7, 418: 7,
408: 7, 406: 7, 402: 7, 395: 7, 394: 7, 383: 7, 378: 7, 377: 7, 375: 7, 368: 7, 361: 7, 351: 7,
341: 7, 329: 7, 327: 7, 312: 7, 304: 7, 291: 7, 284: 7, 267: 7, 264: 7, 932: 6, 823: 6, 800: 6,
784: 6, 761: 6, 741: 6, 708: 6, 700: 6, 696: 6, 638: 6, 606: 6, 600: 6, 591: 6, 586: 6, 576: 6,
567: 6, 557: 6, 544: 6, 535: 6, 533: 6, 529: 6, 523: 6, 518: 6, 498: 6, 490: 6, 489: 6, 486: 6,
480: 6, 473: 6, 456: 6, 450: 6, 438: 6, 437: 6, 433: 6, 432: 6, 430: 6, 426: 6, 410: 6, 391: 6,
386: 6, 384: 6, 382: 6, 379: 6, 349: 6, 335: 6, 332: 6, 330: 6, 289: 6, 2136: 5, 1400: 5, 1117: 5,
989: 5, 930: 5, 883: 5, 853: 5, 851: 5, 824: 5, 818: 5, 810: 5, 771: 5, 770: 5, 769: 5, 763: 5,
719: 5, 703: 5, 701: 5, 683: 5, 674: 5, 671: 5, 662: 5, 637: 5, 636: 5, 633: 5, 630: 5, 628: 5,
619: 5, 610: 5, 603: 5, 589: 5, 585: 5, 580: 5, 577: 5, 574: 5, 571: 5, 569: 5, 554: 5, 553: 5,
551: 5, 538: 5, 530: 5, 528: 5, 527: 5, 516: 5, 508: 5, 500: 5, 495: 5, 485: 5, 476: 5, 472: 5,
423: 5, 422: 5, 409: 5, 398: 5, 388: 5, 372: 5, 342: 5, 290: 5, 1794: 4, 1758: 4, 1507: 4, 1469: 4,
1398: 4, 1351: 4, 1262: 4, 1216: 4, 1194: 4, 1079: 4, 1071: 4, 1065: 4, 1058: 4, 1051: 4, 1032: 4,
1031: 4, 1006: 4, 1001: 4, 964: 4, 958: 4, 957: 4, 894: 4, 888: 4, 887: 4, 881: 4, 874: 4, 866: 4,
856: 4, 847: 4, 812: 4, 785: 4, 782: 4, 775: 4, 751: 4, 750: 4, 742: 4, 735: 4, 733: 4, 732: 4,
730: 4, 728: 4, 727: 4, 715: 4, 694: 4, 693: 4, 687: 4, 686: 4, 684: 4, 682: 4, 680: 4, 678: 4,
664: 4, 663: 4, 658: 4, 656: 4, 651: 4, 650: 4, 649: 4, 645: 4, 642: 4, 635: 4, 632: 4, 631: 4,
625: 4, 620: 4, 618: 4, 613: 4, 597: 4, 595: 4, 592: 4, 584: 4, 583: 4, 578: 4, 572: 4, 568: 4,
560: 4, 556: 4, 552: 4, 545: 4, 521: 4, 519: 4, 517: 4, 512: 4, 510: 4, 506: 4, 505: 4, 503: 4,
497: 4, 488: 4, 483: 4, 482: 4, 478: 4, 475: 4, 471: 4, 470: 4, 468: 4, 465: 4, 464: 4, 461: 4,
457: 4, 455: 4, 454: 4, 453: 4, 451: 4, 448: 4, 439: 4, 428: 4, 427: 4, 420: 4, 413: 4, 412: 4,
401: 4, 400: 4, 392: 4, 389: 4, 380: 4, 353: 4, 321: 4, 316: 4, 2820: 3, 2613: 3, 2468: 3, 2445: 3,
2400: 3, 2270: 3, 2220: 3, 2160: 3, 2066: 3, 2022: 3, 2002: 3, 1977: 3, 1954: 3, 1922: 3, 1920: 3,
1896: 3, 1847: 3, 1821: 3, 1756: 3, 1716: 3, 1670: 3, 1634: 3, 1633: 3, 1630: 3, 1602: 3, 1599: 3,
1536: 3, 1504: 3, 1483: 3, 1454: 3, 1447: 3, 1426: 3, 1410: 3, 1399: 3, 1394: 3, 1391: 3, 1386: 3,
1376: 3, 1362: 3, 1361: 3, 1358: 3, 1344: 3, 1340: 3, 1330: 3, 1324: 3, 1316: 3, 1277: 3, 1270: 3,
1264: 3, 1257: 3, 1250: 3, 1243: 3, 1242: 3, 1241: 3, 1225: 3, 1215: 3, 1196: 3, 1178: 3, 1158: 3,
1151: 3, 1139: 3, 1120: 3, 1116: 3, 1114: 3, 1110: 3, 1108: 3, 1094: 3, 1089: 3, 1074: 3, 1069: 3,
1068: 3, 1064: 3, 1055: 3, 1048: 3, 1046: 3, 1041: 3, 1017: 3, 1004: 3, 995: 3, 986: 3, 965: 3, 951
: 3, 949: 3, 943: 3, 941: 3, 940: 3, 938: 3, 928: 3, 925: 3, 923: 3, 920: 3, 918: 3, 914: 3, 910:
3, 897: 3, 886: 3, 885: 3, 873: 3, 863: 3, 861: 3, 855: 3, 840: 3, 839: 3, 837: 3, 831: 3, 828: 3,
820: 3, 816: 3, 809: 3, 805: 3, 804: 3, 801: 3, 799: 3, 794: 3, 793: 3, 791: 3, 790: 3, 788: 3,
778: 3, 772: 3, 767: 3, 766: 3, 760: 3, 757: 3, 753: 3, 747: 3, 745: 3, 743: 3, 726: 3, 722: 3,
716: 3, 713: 3, 707: 3, 706: 3, 702: 3, 676: 3, 675: 3, 673: 3, 672: 3, 670: 3, 669: 3, 667: 3,
666: 3, 660: 3, 652: 3, 648: 3, 643: 3, 641: 3, 640: 3, 639: 3, 634: 3, 627: 3, 621: 3, 615: 3,
607: 3, 602: 3, 601: 3, 599: 3, 594: 3, 593: 3, 590: 3, 587: 3, 579: 3, 575: 3, 573: 3, 562: 3,
555: 3, 549: 3, 548: 3, 543: 3, 542: 3, 539: 3, 534: 3, 532: 3, 525: 3, 511: 3, 507: 3, 501: 3,
499: 3, 496: 3, 494: 3, 493: 3, 459: 3, 434: 3, 431: 3, 424: 3, 411: 3, 396: 3, 385: 3, 301: 3,
13253: 2, 9431: 2, 6990: 2, 6576: 2, 6058: 2, 5830: 2, 5812: 2, 5311: 2, 5060: 2, 4974: 2, 4620: 2,
4438: 2, 4413: 2, 4383: 2, 4291: 2, 4277: 2, 4229: 2, 4139: 2, 4099: 2, 4090: 2, 4056: 2, 4025: 2,
3929: 2, 3889: 2, 3885: 2, 3807: 2, 3783: 2, 3748: 2, 3678: 2, 3676: 2, 3626: 2, 3583: 2, 3492: 2,
3487: 2, 3485: 2, 3480: 2, 3452: 2, 3371: 2, 3353: 2, 3327: 2, 3315: 2, 3307: 2, 3294: 2, 3281: 2,
3254: 2, 3248: 2, 3162: 2, 3103: 2, 3098: 2, 3049: 2, 2970: 2, 2960: 2, 2867: 2, 2806: 2, 2729: 2,
2719: 2, 2697: 2, 2687: 2, 2662: 2, 2661: 2, 2633: 2, 2631: 2, 2628: 2, 2589: 2, 2588: 2, 2583: 2,
2579: 2, 2574: 2, 2562: 2, 2548: 2, 2544: 2, 2532: 2, 2509: 2, 2475: 2, 2432: 2, 2430: 2, 2417: 2,
2408: 2, 2374: 2, 2357: 2, 2347: 2, 2322: 2, 2315: 2, 2309: 2, 2297: 2, 2296: 2, 2295: 2, 2275: 2,
2263: 2, 2262: 2, 2235: 2, 2209: 2, 2207: 2, 2201: 2, 2182: 2, 2171: 2, 2157: 2, 2148: 2, 2131: 2,
2118: 2, 2101: 2, 2082: 2, 2069: 2, 2031: 2, 2029: 2, 2020: 2, 2018: 2, 2008: 2, 2005: 2, 1995: 2,
1993: 2, 1985: 2, 1960: 2, 1947: 2, 1937: 2, 1931: 2, 1926: 2, 1919: 2, 1916: 2, 1882: 2, 1879: 2,
1868: 2, 1862: 2, 1858: 2, 1857: 2, 1853: 2, 1849: 2, 1836: 2, 1835: 2, 1831: 2, 1830: 2, 1829: 2,
1823: 2, 1815: 2, 1808: 2, 1803: 2, 1798: 2, 1783: 2, 1779: 2, 1775: 2, 1774: 2, 1768: 2, 1767: 2,
1766: 2, 1754: 2, 1753: 2, 1739: 2, 1738: 2, 1734: 2, 1724: 2, 1692: 2, 1691: 2, 1688: 2, 1686: 2,
1677: 2, 1660: 2, 1649: 2, 1644: 2, 1641: 2, 1624: 2, 1622: 2, 1616: 2, 1613: 2, 1611: 2, 1592: 2,
1591: 2, 1580: 2, 1577: 2, 1571: 2, 1570: 2, 1565: 2, 1557: 2, 1555: 2, 1547: 2, 1545: 2, 1541: 2,
1538: 2, 1531: 2, 1510: 2, 1509: 2, 1497: 2, 1490: 2, 1479: 2, 1472: 2, 1467: 2, 1466: 2, 1464: 2,
1451: 2, 1448: 2, 1443: 2, 1441: 2, 1437: 2, 1434: 2, 1417: 2, 1415: 2, 1408: 2, 1404: 2, 1403: 2,
1402: 2, 1401: 2, 1395: 2, 1392: 2, 1385: 2, 1383: 2, 1380: 2, 1378: 2, 1377: 2, 1371: 2, 1370: 2,
1366: 2, 1359: 2, 1357: 2, 1353: 2, 1350: 2, 1345: 2, 1334: 2, 1320: 2, 1311: 2, 1310: 2, 1304: 2,
1297: 2, 1293: 2, 1286: 2, 1284: 2, 1282: 2, 1276: 2, 1273: 2, 1272: 2, 1269: 2, 1267: 2, 1266: 2,
1263: 2, 1253: 2, 1251: 2, 1239: 2, 1235: 2, 1231: 2, 1227: 2, 1224: 2, 1222: 2, 1221: 2, 1219: 2,
1213: 2, 1210: 2, 1209: 2, 1204: 2, 1202: 2, 1199: 2, 1198: 2, 1190: 2, 1185: 2, 1181: 2, 1179: 2,
1176: 2, 1174: 2, 1172: 2, 1170: 2, 1160: 2, 1157: 2, 1152: 2, 1149: 2, 1147: 2, 1143: 2, 1141: 2,
1134: 2, 1128: 2, 1127: 2, 1106: 2, 1099: 2, 1098: 2, 1097: 2, 1095: 2, 1090: 2, 1087: 2, 1084: 2,
1080: 2, 1075: 2, 1073: 2, 1062: 2, 1061: 2, 1054: 2, 1049: 2, 1045: 2, 1043: 2, 1040: 2, 1039: 2,
1037: 2, 1025: 2, 1023: 2, 1022: 2, 1021: 2, 1011: 2, 1010: 2, 1005: 2, 1003: 2, 1002: 2, 993: 2, 9
91: 2, 982: 2, 980: 2, 976: 2, 973: 2, 972: 2, 971: 2, 968: 2, 967: 2, 966: 2, 955: 2, 954: 2,
950: 2, 948: 2, 946: 2, 945: 2, 944: 2, 939: 2, 936: 2, 927: 2, 919: 2, 917: 2, 915: 2, 913: 2,
912: 2, 908: 2, 907: 2, 905: 2, 902: 2, 900: 2, 899: 2, 898: 2, 890: 2, 884: 2, 880: 2, 879: 2,
875: 2, 869: 2, 865: 2, 864: 2, 854: 2, 848: 2, 843: 2, 842: 2, 841: 2, 835: 2, 830: 2, 827: 2,
819: 2, 815: 2, 808: 2, 806: 2, 803: 2, 802: 2, 795: 2, 792: 2, 787: 2, 783: 2, 779: 2, 777: 2,
774: 2, 768: 2, 765: 2, 762: 2, 759: 2, 756: 2, 755: 2, 749: 2, 748: 2, 746: 2, 744: 2, 740: 2,
737: 2, 729: 2, 725: 2, 723: 2, 712: 2, 711: 2, 710: 2, 704: 2, 699: 2, 698: 2, 697: 2, 692: 2,
688: 2, 668: 2, 665: 2, 659: 2, 655: 2, 647: 2, 629: 2, 622: 2, 617: 2, 616: 2, 611: 2, 609: 2,
608: 2, 605: 2, 598: 2, 565: 2, 564: 2, 563: 2, 561: 2, 559: 2, 546: 2, 540: 2, 534: 2, 533: 2,

608: 2, 605: 2, 588: 2, 565: 2, 564: 2, 563: 2, 561: 2, 559: 2, 546: 2, 540: 2, 524: 2, 522: 2,
514: 2, 502: 2, 492: 2, 491: 2, 484: 2, 481: 2, 479: 2, 463: 2, 415: 2, 376: 2, 151608: 1, 120653:
1, 82396: 1, 69427: 1, 69364: 1, 68101: 1, 67144: 1, 64082: 1, 63645: 1, 55223: 1, 55191: 1, 50278
: 1, 49465: 1, 46732: 1, 46601: 1, 43842: 1, 42935: 1, 42910: 1, 42583: 1, 42209: 1, 40812: 1, 405
85: 1, 39184: 1, 38965: 1, 37596: 1, 37430: 1, 36474: 1, 36326: 1, 36249: 1, 34525: 1, 34272: 1, 3
3627: 1, 33431: 1, 33110: 1, 32851: 1, 31926: 1, 29422: 1, 28543: 1, 28237: 1, 27047: 1, 26613: 1,
26219: 1, 26040: 1, 25127: 1, 25096: 1, 24850: 1, 24682: 1, 24647: 1, 24452: 1, 24285: 1, 24213: 1
, 23736: 1, 23460: 1, 22764: 1, 22698: 1, 22113: 1, 21498: 1, 21414: 1, 21029: 1, 21002: 1, 20984:
1, 20751: 1, 20576: 1, 20464: 1, 19948: 1, 19901: 1, 19898: 1, 19663: 1, 19368: 1, 19252: 1, 19110
: 1, 19103: 1, 18769: 1, 18766: 1, 18716: 1, 18393: 1, 18385: 1, 18369: 1, 18342: 1, 18113: 1, 180
78: 1, 17831: 1, 17680: 1, 17668: 1, 17553: 1, 17531: 1, 17410: 1, 17323: 1, 17293: 1, 17116: 1, 1
6983: 1, 16902: 1, 16885: 1, 16858: 1, 16833: 1, 16723: 1, 16639: 1, 16164: 1, 16099: 1, 16046: 1,
15803: 1, 15778: 1, 15714: 1, 15599: 1, 15573: 1, 15481: 1, 15455: 1, 15448: 1, 15291: 1, 15251: 1
, 15225: 1, 15057: 1, 14833: 1, 14807: 1, 14617: 1, 14565: 1, 14495: 1, 14468: 1, 14379: 1, 14342:
1, 14270: 1, 14037: 1, 14016: 1, 13895: 1, 13644: 1, 13617: 1, 13583: 1, 13475: 1, 13466: 1, 13359
: 1, 13280: 1, 13277: 1, 13247: 1, 13187: 1, 13093: 1, 13016: 1, 12980: 1, 12945: 1, 12937: 1, 127
93: 1, 12757: 1, 12677: 1, 12667: 1, 12646: 1, 12569: 1, 12521: 1, 12512: 1, 12494: 1, 12486: 1, 1
2459: 1, 12332: 1, 12312: 1, 12291: 1, 12266: 1, 12245: 1, 12241: 1, 12222: 1, 12180: 1, 12171: 1,
12083: 1, 12072: 1, 12066: 1, 11996: 1, 11986: 1, 11874: 1, 11830: 1, 11766: 1, 11750: 1, 11691: 1
, 11662: 1, 11647: 1, 11610: 1, 11505: 1, 11388: 1, 11354: 1, 11342: 1, 11321: 1, 11276: 1, 11275:
1, 11166: 1, 11139: 1, 10988: 1, 10975: 1, 10912: 1, 10885: 1, 10787: 1, 10717: 1, 10696: 1, 10653
: 1, 10509: 1, 10456: 1, 10447: 1, 10302: 1, 10238: 1, 10201: 1, 10160: 1, 10153: 1, 10152: 1, 101
40: 1, 10125: 1, 10114: 1, 10082: 1, 10075: 1, 10012: 1, 9943: 1, 9930: 1, 9926: 1, 9837: 1, 9751:
1, 9702: 1, 9699: 1, 9653: 1, 9629: 1, 9601: 1, 9589: 1, 9528: 1, 9492: 1, 9480: 1, 9413: 1, 9389:
1, 9365: 1, 9340: 1, 9330: 1, 9321: 1, 9300: 1, 9243: 1, 9093: 1, 9091: 1, 9084: 1, 8997: 1, 8992:
1, 8965: 1, 8963: 1, 8951: 1, 8944: 1, 8924: 1, 8923: 1, 8914: 1, 8900: 1, 8899: 1, 8876: 1, 8869:
1, 8811: 1, 8783: 1, 8701: 1, 8684: 1, 8673: 1, 8655: 1, 8610: 1, 8596: 1, 8589: 1, 8583: 1, 8576:
1, 8435: 1, 8424: 1, 8415: 1, 8339: 1, 8325: 1, 8313: 1, 8306: 1, 8267: 1, 8258: 1, 8236: 1, 8178:
1, 8146: 1, 8145: 1, 8110: 1, 8066: 1, 8058: 1, 8032: 1, 8019: 1, 8009: 1, 8006: 1, 8005: 1, 7948:
1, 7944: 1, 7941: 1, 7939: 1, 7934: 1, 7861: 1, 7850: 1, 7818: 1, 7816: 1, 7806: 1, 7781: 1, 7778:
1, 7777: 1, 7765: 1, 7754: 1, 7749: 1, 7712: 1, 7683: 1, 7668: 1, 7631: 1, 7602: 1, 7597: 1, 7596:
1, 7581: 1, 7535: 1, 7518: 1, 7502: 1, 7498: 1, 7416: 1, 7414: 1, 7402: 1, 7393: 1, 7379: 1, 7286:
1, 7252: 1, 7250: 1, 7234: 1, 7226: 1, 7202: 1, 7183: 1, 7179: 1, 7157: 1, 7149: 1, 7114: 1, 7100:
1, 7094: 1, 7092: 1, 7081: 1, 7068: 1, 7062: 1, 7052: 1, 7040: 1, 7026: 1, 7019: 1, 7006: 1, 6989:
1, 6962: 1, 6953: 1, 6921: 1, 6919: 1, 6893: 1, 6887: 1, 6881: 1, 6876: 1, 6866: 1, 6839: 1, 6793:
1, 6785: 1, 6758: 1, 6756: 1, 6738: 1, 6734: 1, 6730: 1, 6717: 1, 6669: 1, 6668: 1, 6622: 1, 6618:
1, 6610: 1, 6609: 1, 6600: 1, 6591: 1, 6552: 1, 6531: 1, 6530: 1, 6510: 1, 6508: 1, 6482: 1, 6471:
1, 6452: 1, 6434: 1, 6421: 1, 6382: 1, 6372: 1, 6364: 1, 6347: 1, 6342: 1, 6340: 1, 6339: 1, 6337:
1, 6319: 1, 6318: 1, 6315: 1, 6294: 1, 6277: 1, 6260: 1, 6252: 1, 6203: 1, 6184: 1, 6179: 1, 6178:
1, 6168: 1, 6144: 1, 6110: 1, 6097: 1, 6059: 1, 6056: 1, 6053: 1, 6051: 1, 6032: 1, 6026: 1, 5992:
1, 5961: 1, 5953: 1, 5946: 1, 5922: 1, 5883: 1, 5873: 1, 5861: 1, 5857: 1, 5855: 1, 5851: 1, 5843:
1, 5840: 1, 5835: 1, 5832: 1, 5815: 1, 5762: 1, 5748: 1, 5718: 1, 5717: 1, 5709: 1, 5708: 1, 5696:
1, 5693: 1, 5668: 1, 5666: 1, 5651: 1, 5646: 1, 5645: 1, 5625: 1, 5616: 1, 5611: 1, 5608: 1, 5606:
1, 5597: 1, 5557: 1, 5544: 1, 5523: 1, 5519: 1, 5492: 1, 5485: 1, 5478: 1, 5456: 1, 5454: 1, 5445:
1, 5432: 1, 5421: 1, 5395: 1, 5381: 1, 5370: 1, 5367: 1, 5322: 1, 5300: 1, 5278: 1, 5268: 1, 5243:
1, 5240: 1, 5238: 1, 5236: 1, 5235: 1, 5204: 1, 5183: 1, 5140: 1, 5135: 1, 5131: 1, 5103: 1, 5098:
1, 5084: 1, 5081: 1, 5048: 1, 5015: 1, 5010: 1, 5008: 1, 5007: 1, 5001: 1, 4999: 1, 4998: 1, 4997:
1, 4994: 1, 4988: 1, 4964: 1, 4963: 1, 4958: 1, 4952: 1, 4943: 1, 4942: 1, 4939: 1, 4938: 1, 4935:
1, 4931: 1, 4927: 1, 4914: 1, 4907: 1, 4898: 1, 4889: 1, 4880: 1, 4871: 1, 4860: 1, 4858: 1, 4843:
1, 4837: 1, 4827: 1, 4821: 1, 4817: 1, 4813: 1, 4804: 1, 4803: 1, 4799: 1, 4785: 1, 4782: 1, 4776:
1, 4763: 1, 4754: 1, 4753: 1, 4728: 1, 4726: 1, 4714: 1, 4676: 1, 4670: 1, 4667: 1, 4664: 1, 4663:
1, 4660: 1, 4635: 1, 4625: 1, 4615: 1, 4612: 1, 4604: 1, 4602: 1, 4599: 1, 4597: 1, 4596: 1, 4582:
1, 4580: 1, 4568: 1, 4567: 1, 4542: 1, 4541: 1, 4540: 1, 4532: 1, 4528: 1, 4513: 1, 4492: 1, 4480:
1, 4465: 1, 4436: 1, 4421: 1, 4420: 1, 4416: 1, 4401: 1, 4388: 1, 4381: 1, 4365: 1, 4347: 1, 4335:
1, 4327: 1, 4326: 1, 4322: 1, 4320: 1, 4310: 1, 4300: 1, 4293: 1, 4292: 1, 4288: 1, 4282: 1, 4272:
1, 4269: 1, 4264: 1, 4262: 1, 4261: 1, 4254: 1, 4244: 1, 4243: 1, 4223: 1, 4221: 1, 4219: 1, 4213:
1, 4205: 1, 4197: 1, 4195: 1, 4187: 1, 4177: 1, 4175: 1, 4166: 1, 4164: 1, 4163: 1, 4152: 1, 4151:
1, 4150: 1, 4129: 1, 4128: 1, 4127: 1, 4125: 1, 4122: 1, 4093: 1, 4087: 1, 4085: 1, 4069: 1, 4060:
1, 4031: 1, 4017: 1, 4003: 1, 3990: 1, 3989: 1, 3985: 1, 3984: 1, 3982: 1, 3959: 1, 3957: 1, 3939:
1, 3931: 1, 3927: 1, 3924: 1, 3914: 1, 3911: 1, 3908: 1, 3906: 1, 3905: 1, 3886: 1, 3875: 1, 3873:
1, 3864: 1, 3858: 1, 3834: 1, 3829: 1, 3826: 1, 3820: 1, 3814: 1, 3813: 1, 3811: 1, 3809: 1, 3804:
1, 3799: 1, 3797: 1, 3794: 1, 3785: 1, 3781: 1, 3778: 1, 3777: 1, 3774: 1, 3768: 1, 3765: 1, 3764:
1, 3758: 1, 3754: 1, 3746: 1, 3744: 1, 3734: 1, 3720: 1, 3709: 1, 3700: 1, 3698: 1, 3693: 1, 3688:
1, 3683: 1, 3672: 1, 3662: 1, 3661: 1, 3646: 1, 3637: 1, 3632: 1, 3630: 1, 3621: 1, 3613: 1, 3598:
1, 3593: 1, 3590: 1, 3586: 1, 3578: 1, 3575: 1, 3571: 1, 3568: 1, 3565: 1, 3563: 1, 3560: 1, 3559:
1, 3553: 1, 3551: 1, 3541: 1, 3538: 1, 3537: 1, 3533: 1, 3531: 1, 3529: 1, 3528: 1, 3523: 1, 3522:
1, 3521: 1, 3518: 1, 3512: 1, 3501: 1, 3493: 1, 3491: 1, 3486: 1, 3481: 1, 3477: 1, 3469: 1, 3464:
1, 3460: 1, 3459: 1, 3451: 1, 3447: 1, 3443: 1, 3439: 1, 3433: 1, 3423: 1, 3421: 1, 3418: 1, 3410:
1, 3407: 1, 3402: 1, 3399: 1, 3397: 1, 3373: 1, 3372: 1, 3368: 1, 3367: 1, 3366: 1, 3365: 1, 3355:
1, 3350: 1, 3343: 1, 3340: 1, 3338: 1, 3334: 1, 3331: 1, 3330: 1, 3329: 1, 3314: 1, 3306: 1, 3304:
1, 3289: 1, 3287: 1, 3284: 1, 3277: 1, 3270: 1, 3262: 1, 3258: 1, 3255: 1, 3237: 1, 3236: 1, 3234:
1, 3233: 1, 3232: 1, 3226: 1, 3225: 1, 3219: 1, 3216: 1, 3205: 1, 3204: 1, 3202: 1, 3192: 1, 3191:
1, 3189: 1, 3183: 1, 3179: 1, 3169: 1, 3165: 1, 3161: 1, 3156: 1, 3154: 1, 3150: 1, 3147: 1, 3141:
1, 3138: 1, 3135: 1, 3132: 1, 3124: 1, 3122: 1, 3121: 1, 3119: 1, 3115: 1, 3113: 1, 3112: 1, 3109:
1, 3106: 1, 3096: 1, 3094: 1, 3090: 1, 3081: 1, 3080: 1, 3078: 1, 3073: 1, 3072: 1, 3069: 1, 3068:
1, 3066: 1, 3057: 1, 3055: 1, 3054: 1, 3050: 1, 3040: 1, 3038: 1, 3034: 1, 3021: 1, 3017: 1, 3013:
1, 3012: 1, 2992: 1, 2987: 1, 2985: 1, 2983: 1, 2980: 1, 2978: 1, 2968: 1, 2966: 1, 2956: 1, 2955:

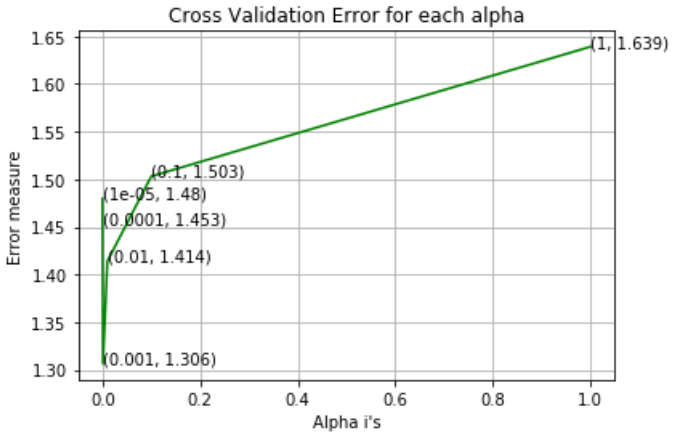
```
1, 2946: 1, 2940: 1, 2925: 1, 2920: 1, 2918: 1, 2905: 1, 2903: 1, 2900: 1, 2891: 1, 2890: 1, 2889:
1, 2885: 1, 2884: 1, 2875: 1, 2874: 1, 2864: 1, 2862: 1, 2858: 1, 2852: 1, 2850: 1, 2848: 1, 2841:
1, 2835: 1, 2823: 1, 2822: 1, 2815: 1, 2801: 1, 2797: 1, 2793: 1, 2790: 1, 2789: 1, 2781: 1, 2780:
1, 2779: 1, 2778: 1, 2773: 1, 2767: 1, 2763: 1, 2762: 1, 2761: 1, 2754: 1, 2740: 1, 2733: 1, 2732:
1, 2727: 1, 2724: 1, 2723: 1, 2708: 1, 2704: 1, 2696: 1, 2688: 1, 2678: 1, 2673: 1, 2672: 1, 2670:
1, 2666: 1, 2664: 1, 2659: 1, 2656: 1, 2652: 1, 2648: 1, 2647: 1, 2646: 1, 2644: 1, 2641: 1, 2640:
1, 2635: 1, 2627: 1, 2622: 1, 2617: 1, 2614: 1, 2611: 1, 2604: 1, 2591: 1, 2586: 1, 2581: 1, 2580:
1, 2577: 1, 2571: 1, 2569: 1, 2568: 1, 2566: 1, 2565: 1, 2560: 1, 2554: 1, 2553: 1, 2537: 1, 2535:
1, 2534: 1, 2531: 1, 2528: 1, 2522: 1, 2517: 1, 2516: 1, 2514: 1, 2508: 1, 2505: 1, 2498: 1, 2496:
1, 2489: 1, 2487: 1, 2486: 1, 2478: 1, 2477: 1, 2471: 1, 2464: 1, 2463: 1, 2456: 1, 2450: 1, 2446:
1, 2436: 1, 2431: 1, 2426: 1, 2425: 1, 2421: 1, 2419: 1, 2413: 1, 2411: 1, 2404: 1, 2401: 1, 2396:
1, 2395: 1, 2393: 1, 2392: 1, 2391: 1, 2390: 1, 2387: 1, 2380: 1, 2371: 1, 2369: 1, 2367: 1, 2366:
1, 2365: 1, 2363: 1, 2361: 1, 2358: 1, 2355: 1, 2342: 1, 2336: 1, 2332: 1, 2329: 1, 2326: 1, 2325:
1, 2310: 1, 2306: 1, 2300: 1, 2282: 1, 2278: 1, 2271: 1, 2265: 1, 2261: 1, 2259: 1, 2255: 1, 2254:
1, 2251: 1, 2250: 1, 2249: 1, 2247: 1, 2241: 1, 2240: 1, 2239: 1, 2233: 1, 2229: 1, 2227: 1, 2224:
1, 2222: 1, 2221: 1, 2216: 1, 2212: 1, 2206: 1, 2203: 1, 2199: 1, 2197: 1, 2193: 1, 2191: 1, 2190:
1, 2188: 1, 2186: 1, 2184: 1, 2177: 1, 2175: 1, 2166: 1, 2153: 1, 2151: 1, 2149: 1, 2144: 1, 2142:
1, 2139: 1, 2133: 1, 2130: 1, 2127: 1, 2126: 1, 2125: 1, 2124: 1, 2121: 1, 2120: 1, 2119: 1, 2117:
1, 2115: 1, 2113: 1, 2111: 1, 2103: 1, 2100: 1, 2099: 1, 2095: 1, 2089: 1, 2088: 1, 2086: 1, 2085:
1, 2084: 1, 2080: 1, 2075: 1, 2073: 1, 2070: 1, 2067: 1, 2063: 1, 2056: 1, 2055: 1, 2048: 1, 2047:
1, 2045: 1, 2043: 1, 2039: 1, 2037: 1, 2035: 1, 2026: 1, 2025: 1, 2017: 1, 2016: 1, 2015: 1, 2014:
1, 2013: 1, 2012: 1, 2011: 1, 2001: 1, 1988: 1, 1986: 1, 1983: 1, 1982: 1, 1976: 1, 1971: 1, 1970:
1, 1969: 1, 1968: 1, 1966: 1, 1964: 1, 1963: 1, 1957: 1, 1955: 1, 1950: 1, 1948: 1, 1945: 1, 1944:
1, 1943: 1, 1941: 1, 1934: 1, 1932: 1, 1929: 1, 1928: 1, 1927: 1, 1917: 1, 1910: 1, 1909: 1, 1908:
1, 1906: 1, 1902: 1, 1901: 1, 1900: 1, 1899: 1, 1895: 1, 1893: 1, 1890: 1, 1888: 1, 1885: 1, 1884:
1, 1881: 1, 1880: 1, 1877: 1, 1873: 1, 1872: 1, 1859: 1, 1852: 1, 1848: 1, 1846: 1, 1841: 1, 1840:
1, 1832: 1, 1824: 1, 1818: 1, 1816: 1, 1810: 1, 1807: 1, 1806: 1, 1805: 1, 1801: 1, 1800: 1, 1795:
1, 1792: 1, 1789: 1, 1785: 1, 1777: 1, 1776: 1, 1773: 1, 1771: 1, 1769: 1, 1765: 1, 1763: 1, 1761:
1, 1759: 1, 1757: 1, 1752: 1, 1750: 1, 1747: 1, 1746: 1, 1741: 1, 1736: 1, 1735: 1, 1730: 1, 1729:
1, 1725: 1, 1723: 1, 1718: 1, 1717: 1, 1714: 1, 1711: 1, 1707: 1, 1706: 1, 1702: 1, 1700: 1, 1699:
1, 1698: 1, 1696: 1, 1694: 1, 1693: 1, 1690: 1, 1687: 1, 1684: 1, 1681: 1, 1679: 1, 1676: 1, 1674:
1, 1671: 1, 1668: 1, 1666: 1, 1661: 1, 1659: 1, 1657: 1, 1651: 1, 1650: 1, 1646: 1, 1645: 1, 1643:
1, 1640: 1, 1639: 1, 1638: 1, 1637: 1, 1636: 1, 1632: 1, 1631: 1, 1623: 1, 1619: 1, 1618: 1, 1617:
1, 1614: 1, 1612: 1, 1610: 1, 1609: 1, 1608: 1, 1607: 1, 1606: 1, 1605: 1, 1601: 1, 1600: 1, 1597:
1, 1596: 1, 1588: 1, 1587: 1, 1584: 1, 1582: 1, 1581: 1, 1576: 1, 1575: 1, 1572: 1, 1566: 1, 1564:
1, 1561: 1, 1556: 1, 1551: 1, 1549: 1, 1546: 1, 1544: 1, 1542: 1, 1539: 1, 1537: 1, 1535: 1, 1530:
1, 1527: 1, 1525: 1, 1523: 1, 1520: 1, 1519: 1, 1515: 1, 1514: 1, 1513: 1, 1512: 1, 1511: 1, 1503:
1, 1500: 1, 1499: 1, 1496: 1, 1495: 1, 1493: 1, 1491: 1, 1488: 1, 1485: 1, 1482: 1, 1476: 1, 1473:
1, 1470: 1, 1465: 1, 1462: 1, 1457: 1, 1456: 1, 1455: 1, 1452: 1, 1446: 1, 1445: 1, 1442: 1, 1440:
1, 1439: 1, 1436: 1, 1435: 1, 1432: 1, 1429: 1, 1428: 1, 1427: 1, 1425: 1, 1424: 1, 1422: 1, 1421:
1, 1420: 1, 1419: 1, 1418: 1, 1414: 1, 1412: 1, 1411: 1, 1409: 1, 1406: 1, 1396: 1, 1393: 1, 1388:
1, 1387: 1, 1384: 1, 1382: 1, 1381: 1, 1375: 1, 1373: 1, 1372: 1, 1369: 1, 1365: 1, 1364: 1, 1356:
1, 1354: 1, 1349: 1, 1347: 1, 1343: 1, 1342: 1, 1341: 1, 1339: 1, 1337: 1, 1335: 1, 1333: 1, 1332:
1, 1327: 1, 1326: 1, 1325: 1, 1322: 1, 1321: 1, 1319: 1, 1318: 1, 1317: 1, 1315: 1, 1314: 1, 1312:
1, 1308: 1, 1303: 1, 1302: 1, 1301: 1, 1299: 1, 1296: 1, 1295: 1, 1294: 1, 1291: 1, 1290: 1, 1285:
1, 1283: 1, 1280: 1, 1278: 1, 1271: 1, 1268: 1, 1265: 1, 1260: 1, 1259: 1, 1258: 1, 1249: 1, 1247:
1, 1245: 1, 1240: 1, 1238: 1, 1232: 1, 1229: 1, 1228: 1, 1226: 1, 1223: 1, 1217: 1, 1214: 1, 1212:
1, 1208: 1, 1207: 1, 1206: 1, 1205: 1, 1200: 1, 1195: 1, 1193: 1, 1192: 1, 1189: 1, 1188: 1, 1186:
1, 1182: 1, 1173: 1, 1171: 1, 1169: 1, 1168: 1, 1167: 1, 1165: 1, 1164: 1, 1156: 1, 1155: 1, 1150:
1, 1148: 1, 1142: 1, 1140: 1, 1137: 1, 1136: 1, 1135: 1, 1132: 1, 1130: 1, 1129: 1, 1126: 1, 1123:
1, 1122: 1, 1118: 1, 1113: 1, 1107: 1, 1105: 1, 1104: 1, 1103: 1, 1100: 1, 1096: 1, 1093: 1, 1091:
1, 1082: 1, 1077: 1, 1076: 1, 1072: 1, 1070: 1, 1067: 1, 1063: 1, 1059: 1, 1056: 1, 1053: 1, 1050:
1, 1044: 1, 1038: 1, 1036: 1, 1035: 1, 1030: 1, 1029: 1, 1028: 1, 1027: 1, 1026: 1, 1024: 1, 1018:
1, 1016: 1, 1015: 1, 1014: 1, 1013: 1, 1012: 1, 1009: 1, 1008: 1, 1007: 1, 1000: 1, 999: 1, 998: 1,
997: 1, 996: 1, 994: 1, 992: 1, 990: 1, 988: 1, 985: 1, 983: 1, 979: 1, 978: 1, 977: 1, 970: 1,
963: 1, 962: 1, 960: 1, 959: 1, 956: 1, 953: 1, 952: 1, 947: 1, 937: 1, 935: 1, 934: 1, 933: 1,
926: 1, 924: 1, 921: 1, 916: 1, 911: 1, 909: 1, 904: 1, 903: 1, 901: 1, 895: 1, 892: 1, 891: 1,
889: 1, 878: 1, 877: 1, 876: 1, 872: 1, 871: 1, 870: 1, 867: 1, 862: 1, 858: 1, 857: 1, 852: 1,
849: 1, 844: 1, 838: 1, 836: 1, 834: 1, 833: 1, 832: 1, 826: 1, 825: 1, 817: 1, 811: 1, 807: 1,
798: 1, 797: 1, 796: 1, 786: 1, 781: 1, 776: 1, 773: 1, 764: 1, 752: 1, 739: 1, 738: 1, 736: 1,
734: 1, 731: 1, 721: 1, 718: 1, 695: 1, 691: 1, 690: 1, 689: 1, 685: 1, 681: 1, 677: 1, 661: 1,
624: 1, 623: 1, 614: 1, 612: 1, 604: 1, 598: 1, 596: 1, 582: 1, 581: 1, 558: 1, 526: 1, 487: 1,
356: 1})
```

In [0]:

```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
```

```
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#--------------------------------
# video link:
#--------------------------------


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    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)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
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_text_feature_onehotCoding)
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_text_feature_onehotCoding)
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: 1.4798785760828574
For values of alpha =  0.0001 The log loss is: 1.4534937808292785
For values of alpha =  0.001 The log loss is: 1.3062465612808916
For values of alpha =  0.01 The log loss is: 1.4138930975129917
For values of alpha =  0.1 The log loss is: 1.5031084843525189
For values of alpha =  1 The log loss is: 1.6391636597237738
```



```
For values of best alpha =  0.001 The train log loss is: 0.7614133457365512
For values of best alpha =  0.001 The cross validation log loss is: 1.3062465612808916
For values of best alpha =  0.001 The test log loss is: 1.1902669954542044
```

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

Ans. Yes, it seems like!

In [0]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    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 [0]:

```
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")
```

```
97.125 % of word of test data appeared in train data
98.056 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [13]:

```
#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 [14]:

```
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 [15]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
```

```
    text_count_vec = CountVectorizer(min_df=3)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_r
o))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

    print("Out of the top ",no_features," features ", word_present, "are present in query point")
```

Stacking the three types of features

In [34]:

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)
)

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
r()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding =
np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding =
np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
cv_gene_var_responseCoding =
np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,
```

```
train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding)
)
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
```

In [0]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding
.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 57039)
(number of data points * number of features) in test data =  (665, 57039)
(number of data points * number of features) in cross validation data = (532, 57039)
```

In [0]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shap
e)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

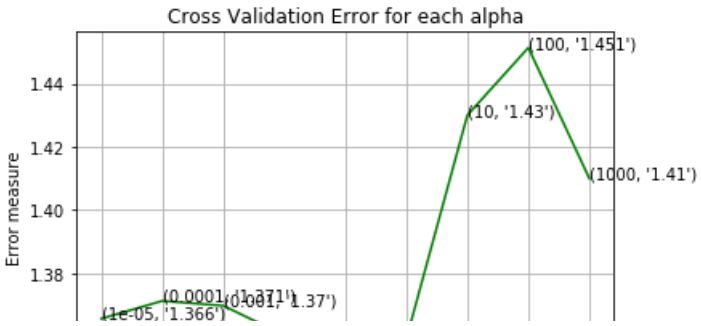
4.1.1.1. Hyper parameter tuning

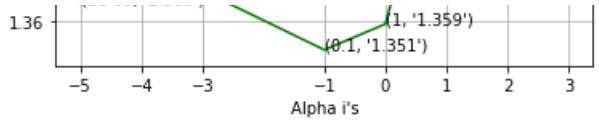
In [0]:

```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-
algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# ----------------------------
```

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# ------------------------


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    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(np.log10(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)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
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 = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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_x_onehotCoding)
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_x_onehotCoding)
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 = 1e-05
Log Loss : 1.3655537837941127
for alpha = 0.0001
Log Loss : 1.3711793514758466
for alpha = 0.001
Log Loss : 1.3696542195047903
for alpha = 0.1
Log Loss : 1.3509235125982695
for alpha = 1
Log Loss : 1.3591155403248767
for alpha = 10
Log Loss : 1.4299766791532638
for alpha = 100
Log Loss : 1.451360452876549
for alpha = 1000
Log Loss : 1.4099515277732073
```





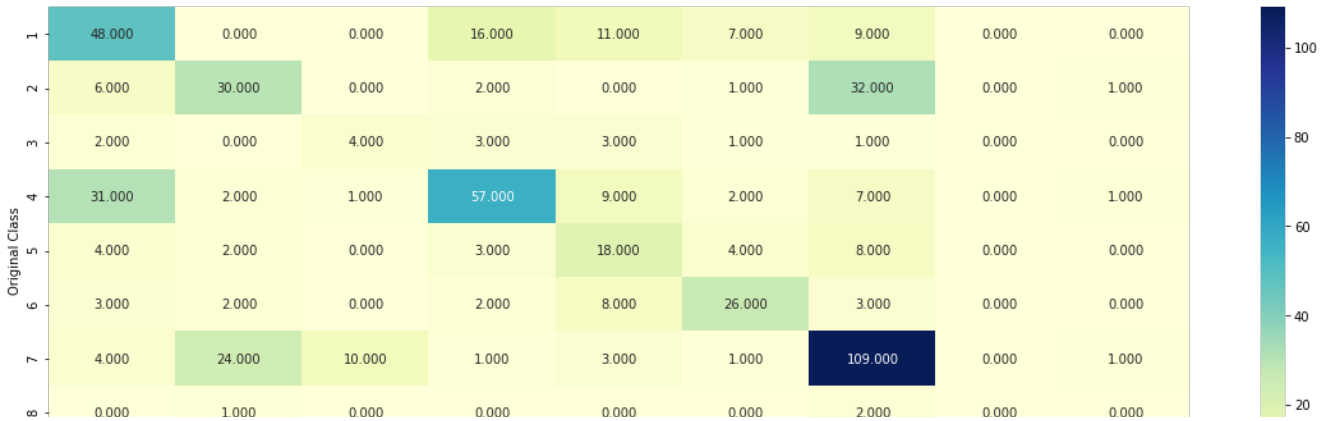

```
For values of best alpha =  0.1 The train log loss is: 0.9033118479519152
For values of best alpha =  0.1 The cross validation log loss is: 1.3509235125982695
For values of best alpha =  0.1 The test log loss is: 1.2784897724659714
```

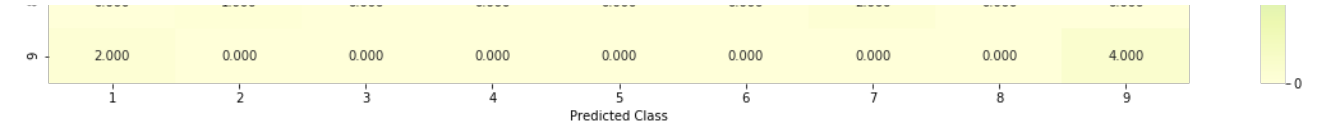
4.1.1.2. Testing the model with best hyper paramters

In [0]:

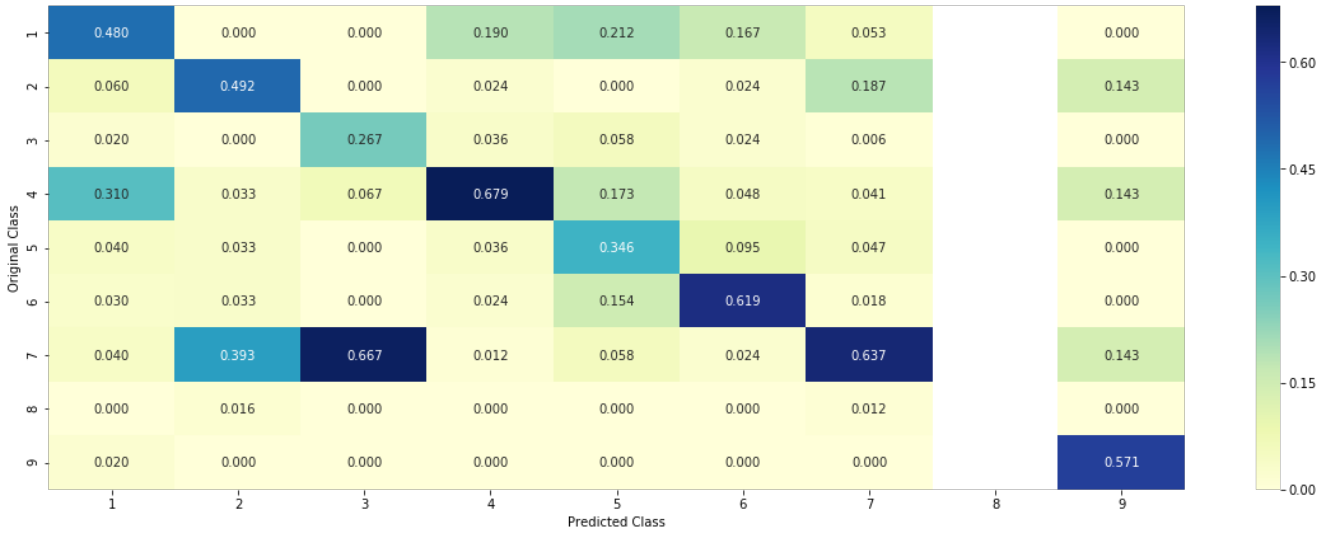
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-
algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# ----------------------------

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv
_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.3509235125982695
Number of missclassified point : 0.44360902255639095
-------------------- Confusion matrix --------------------
```

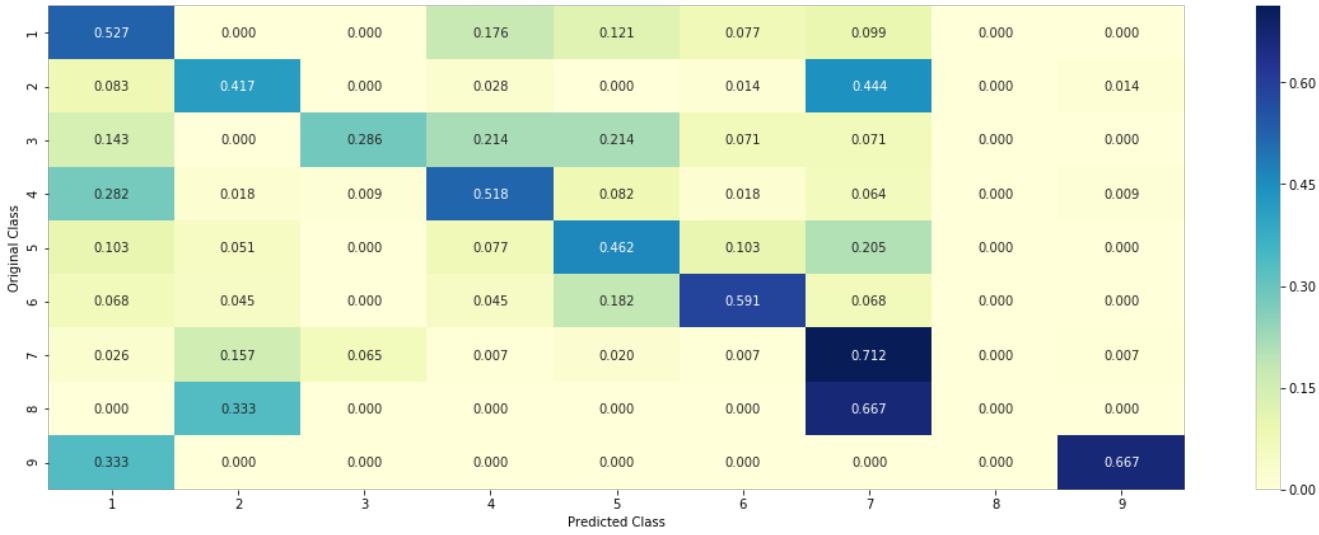





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.1.1.3. Feature Importance, Correctly classified point

In [0]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0911 0.1284 0.0141 0.1119 0.0334 0.0365 0.5764 0.0057 0.0025]]
Actual Class : 7
```

```
--------------------------------------------------
16 Text feature [presence] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [well] present in test data point [True]
19 Text feature [activating] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [cell] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [independent] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [recently] present in test data point [True]
29 Text feature [shown] present in test data point [True]
30 Text feature [potential] present in test data point [True]
31 Text feature [also] present in test data point [True]
32 Text feature [obtained] present in test data point [True]
33 Text feature [growth] present in test data point [True]
34 Text feature [activation] present in test data point [True]
35 Text feature [suggest] present in test data point [True]
36 Text feature [showed] present in test data point [True]
37 Text feature [however] present in test data point [True]
38 Text feature [expressing] present in test data point [True]
39 Text feature [addition] present in test data point [True]
40 Text feature [found] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [previously] present in test data point [True]
43 Text feature [factor] present in test data point [True]
44 Text feature [compared] present in test data point [True]
45 Text feature [treated] present in test data point [True]
46 Text feature [inhibition] present in test data point [True]
47 Text feature [higher] present in test data point [True]
48 Text feature [observed] present in test data point [True]
49 Text feature [described] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [similar] present in test data point [True]
52 Text feature [total] present in test data point [True]
53 Text feature [furthermore] present in test data point [True]
54 Text feature [studies] present in test data point [True]
55 Text feature [using] present in test data point [True]
56 Text feature [without] present in test data point [True]
57 Text feature [concentrations] present in test data point [True]
58 Text feature [1a] present in test data point [True]
59 Text feature [various] present in test data point [True]
60 Text feature [including] present in test data point [True]
61 Text feature [mutations] present in test data point [True]
62 Text feature [respectively] present in test data point [True]
63 Text feature [12] present in test data point [True]
64 Text feature [followed] present in test data point [True]
65 Text feature [enhanced] present in test data point [True]
66 Text feature [although] present in test data point [True]
67 Text feature [interestingly] present in test data point [True]
68 Text feature [phosphorylation] present in test data point [True]
70 Text feature [new] present in test data point [True]
71 Text feature [inhibited] present in test data point [True]
72 Text feature [constitutively] present in test data point [True]
75 Text feature [1b] present in test data point [True]
76 Text feature [reported] present in test data point [True]
77 Text feature [confirmed] present in test data point [True]
78 Text feature [inhibitors] present in test data point [True]
79 Text feature [proliferation] present in test data point [True]
80 Text feature [report] present in test data point [True]
81 Text feature [either] present in test data point [True]
82 Text feature [molecular] present in test data point [True]
83 Text feature [15] present in test data point [True]
84 Text feature [thus] present in test data point [True]
85 Text feature [recent] present in test data point [True]
86 Text feature [3b] present in test data point [True]
87 Text feature [fig] present in test data point [True]
88 Text feature [results] present in test data point [True]
89 Text feature [occur] present in test data point [True]
90 Text feature [small] present in test data point [True]
91 Text feature [3a] present in test data point [True]
92 Text feature [approximately] present in test data point [True]
93 Text feature [hours] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [figure] present in test data point [True]
97 Text feature [suggests] present in test data point [True]
```

```
98 Text feature [absence] present in test data point [True]
99 Text feature [measured] present in test data point [True]
Out of the top  100  features  78 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [0]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0876 0.0895 0.0136 0.1069 0.0322 0.0356 0.6267 0.0055 0.0024]]
Actual Class : 7
--------------------------------------------------
17 Text feature [kinase] present in test data point [True]
18 Text feature [well] present in test data point [True]
19 Text feature [activating] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [cell] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [independent] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [recently] present in test data point [True]
29 Text feature [shown] present in test data point [True]
30 Text feature [potential] present in test data point [True]
31 Text feature [also] present in test data point [True]
33 Text feature [growth] present in test data point [True]
34 Text feature [activation] present in test data point [True]
35 Text feature [suggest] present in test data point [True]
36 Text feature [showed] present in test data point [True]
37 Text feature [however] present in test data point [True]
39 Text feature [addition] present in test data point [True]
40 Text feature [found] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [previously] present in test data point [True]
44 Text feature [compared] present in test data point [True]
46 Text feature [inhibition] present in test data point [True]
47 Text feature [higher] present in test data point [True]
48 Text feature [observed] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [similar] present in test data point [True]
54 Text feature [studies] present in test data point [True]
55 Text feature [using] present in test data point [True]
56 Text feature [without] present in test data point [True]
58 Text feature [1a] present in test data point [True]
60 Text feature [including] present in test data point [True]
61 Text feature [mutations] present in test data point [True]
62 Text feature [respectively] present in test data point [True]
63 Text feature [12] present in test data point [True]
64 Text feature [followed] present in test data point [True]
65 Text feature [enhanced] present in test data point [True]
66 Text feature [although] present in test data point [True]
68 Text feature [phosphorylation] present in test data point [True]
69 Text feature [activated] present in test data point [True]
70 Text feature [new] present in test data point [True]
71 Text feature [inhibited] present in test data point [True]
72 Text feature [constitutively] present in test data point [True]
75 Text feature [1b] present in test data point [True]
78 Text feature [inhibitors] present in test data point [True]
79 Text feature [proliferation] present in test data point [True]
81 Text feature [either] present in test data point [True]
82 Text feature [molecular] present in test data point [True]
83 Text feature [15] present in test data point [True]
```

```
85 Text feature [recent] present in test data point [True]
86 Text feature [3b] present in test data point [True]
87 Text feature [fig] present in test data point [True]
88 Text feature [results] present in test data point [True]
89 Text feature [occur] present in test data point [True]
90 Text feature [small] present in test data point [True]
91 Text feature [3a] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [figure] present in test data point [True]
97 Text feature [suggests] present in test data point [True]
98 Text feature [absence] present in test data point [True]
Out of the top  100  features  60 are present in query point
```

4.2. K Nearest Neighbour Classification

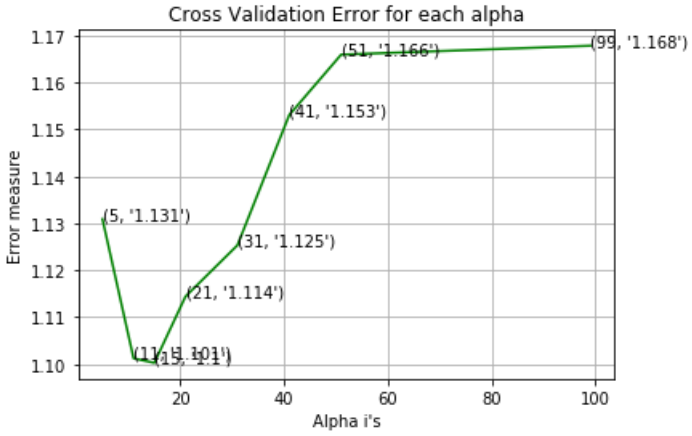
4.2.1. Hyper parameter tuning

In [0]:

```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------


alpha = [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_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    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_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
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_x_responseCoding)
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_x_responseCoding)
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 : 1.1309170126692265
for alpha = 11
Log Loss : 1.1012397762291362
for alpha = 15
Log Loss : 1.1002748803755749
for alpha = 21
Log Loss : 1.1144110925957647
for alpha = 31
Log Loss : 1.1253206995500455
for alpha = 41
Log Loss : 1.1530939773909168
for alpha = 51
Log Loss : 1.1659585643007098
for alpha = 99
Log Loss : 1.1678505034822115
```




```
For values of best alpha =  15 The train log loss is: 0.7056892871225193
For values of best alpha =  15 The cross validation log loss is: 1.1002748803755749
For values of best alpha =  15 The test log loss is: 1.0911901980302394
```

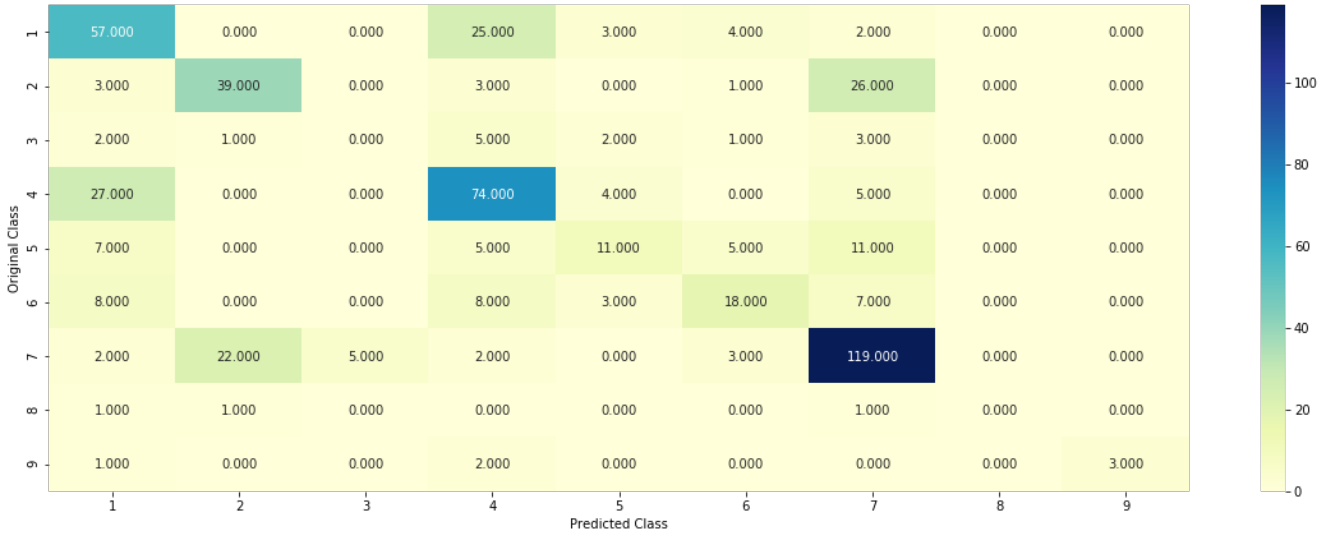
4.2.2. Testing the model with best hyper paramters

In [0]:

```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
```

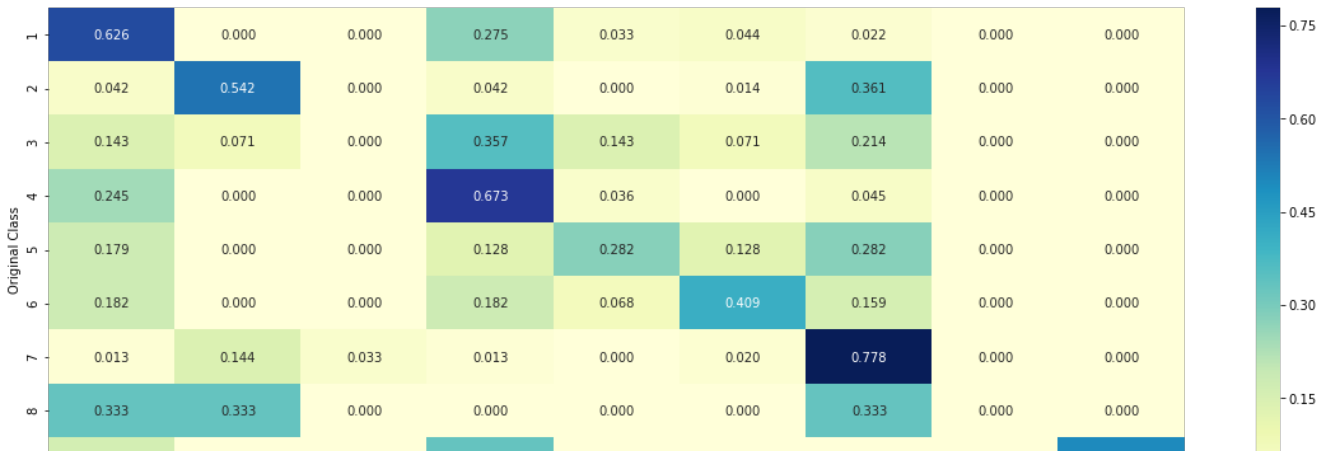
```
"
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

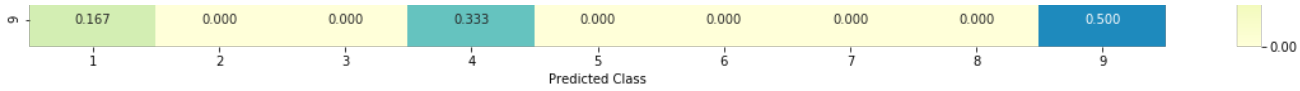
```
Log loss : 1.1002748803755749
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```


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





4.2.3.Sample Query point -1

In [0]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha
])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y
[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 6
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [7 7 7 6 6 7 6 7 6 7 7 7 7 7 6]
Fequency of nearest points : Counter({7: 10, 6: 5})
```

4.2.4. Sample Query Point-2

In [0]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha
])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test points be
longs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [2 7 5
7 7 2 7 7 7 2 7 7 7 7 7]
Fequency of nearest points : Counter({7: 11, 2: 3, 5: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

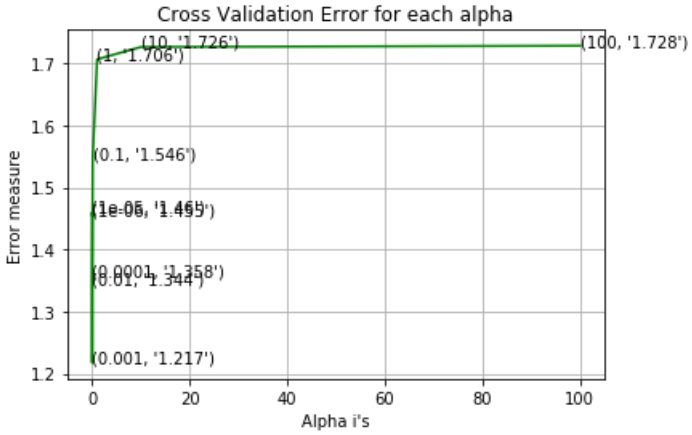
In [0]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
```

```
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#--------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#--------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    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 = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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_x_onehotCoding)
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_x_onehotCoding)
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 = 1e-06
Log Loss : 1.4554353198842396
for alpha = 1e-05
```

```
Log Loss : 1.4602144866667575
for alpha = 0.0001
Log Loss : 1.358469527280309
for alpha = 0.001
Log Loss : 1.217324457704446
for alpha = 0.01
Log Loss : 1.3437838209291793
for alpha = 0.1
Log Loss : 1.5457557924182381
for alpha = 1
Log Loss : 1.706360520395438
for alpha = 10
Log Loss : 1.7261917214695601
for alpha = 100
Log Loss : 1.7282505302427342
```



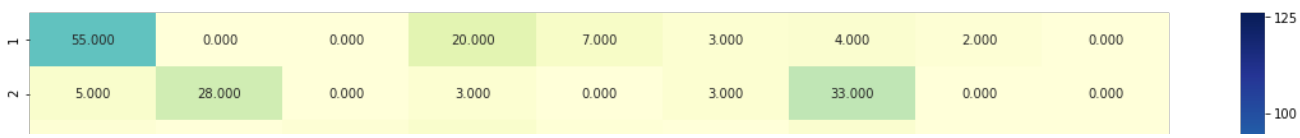

```
For values of best alpha =  0.001 The train log loss is: 0.6153177097029675
For values of best alpha =  0.001 The cross validation log loss is: 1.217324457704446
For values of best alpha =  0.001 The test log loss is: 1.1047257743181136
```

4.3.1.2. Testing the model with best hyper paramters

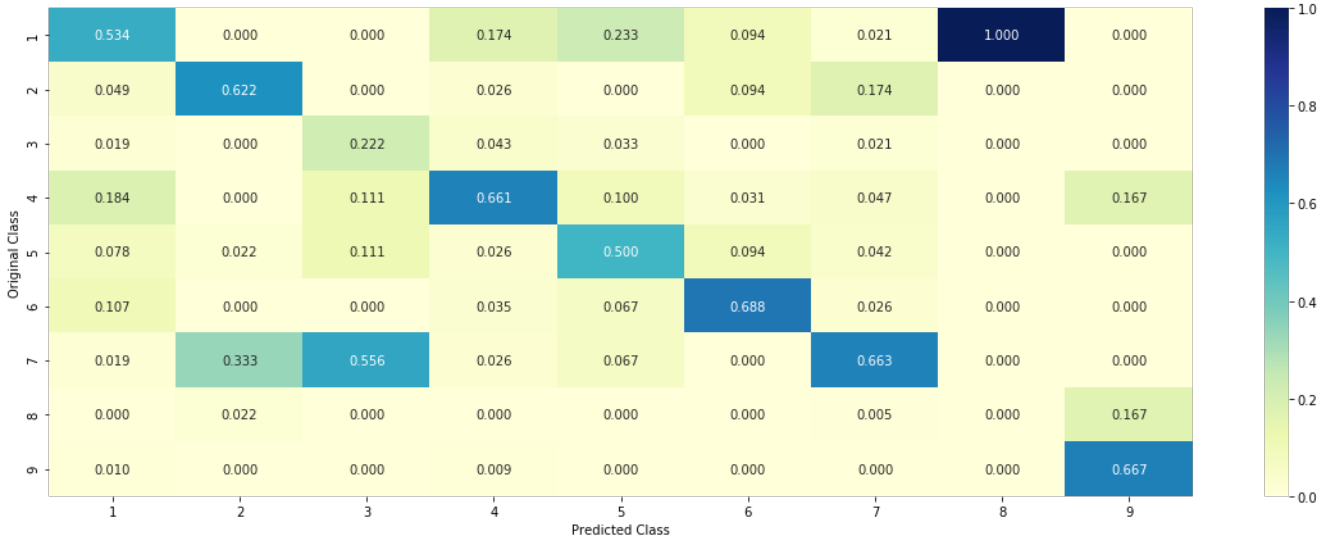
In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

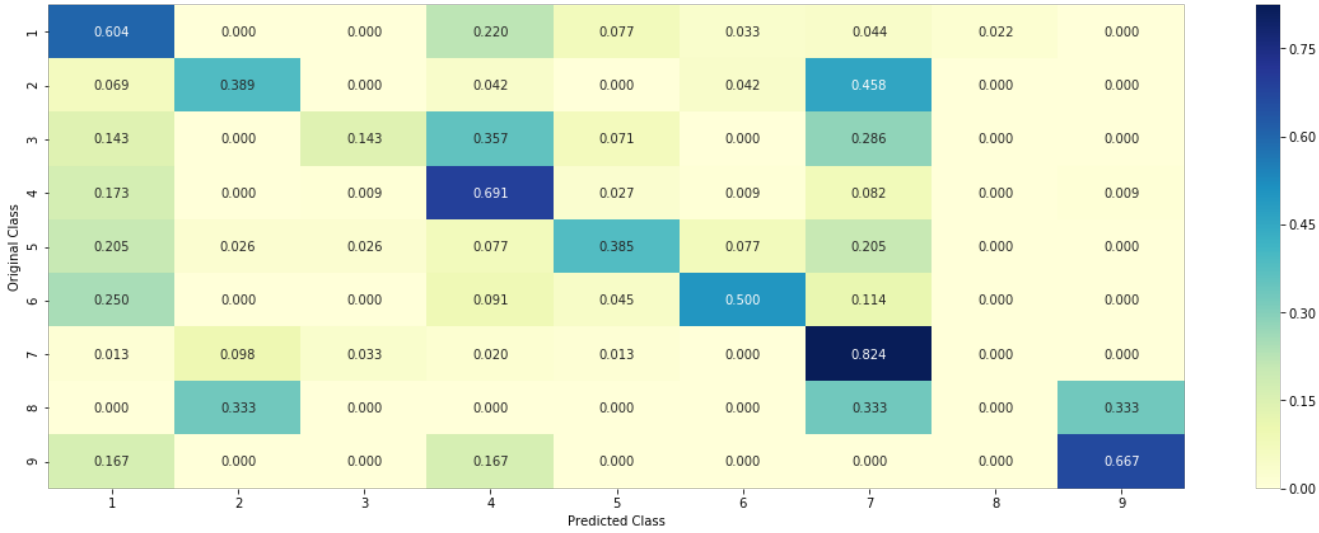
```
Log loss : 1.217324457704446
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```



---------- Precision matrix (Columm Sum=1) ----------



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



4.3.1.3. Feature Importance

In [0]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
```

```
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [0]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0045 0.1905 0.0012 0.0012 0.0047 0.0014 0.7872 0.0076 0.0017]]
Actual Class : 7
--------------------------------------------------
23 Text feature [constitutively] present in test data point [True]
39 Text feature [flt1] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
80 Text feature [oncogenes] present in test data point [True]
84 Text feature [cysteine] present in test data point [True]
89 Text feature [inhibited] present in test data point [True]
137 Text feature [technology] present in test data point [True]
160 Text feature [dramatic] present in test data point [True]
162 Text feature [gaiix] present in test data point [True]
166 Text feature [ligand] present in test data point [True]
177 Text feature [downstream] present in test data point [True]
181 Text feature [concentrations] present in test data point [True]
182 Text feature [thyroid] present in test data point [True]
187 Text feature [expressing] present in test data point [True]
217 Text feature [activating] present in test data point [True]
241 Text feature [cdnas] present in test data point [True]
250 Text feature [manageable] present in test data point [True]
265 Text feature [axilla] present in test data point [True]
302 Text feature [inhibitor] present in test data point [True]
311 Text feature [cot] present in test data point [True]
313 Text feature [viability] present in test data point [True]
334 Text feature [activation] present in test data point [True]
352 Text feature [forced] present in test data point [True]
368 Text feature [subcutaneous] present in test data point [True]
371 Text feature [melanocyte] present in test data point [True]
376 Text feature [erk1] present in test data point [True]
388 Text feature [hours] present in test data point [True]
446 Text feature [procure] present in test data point [True]
448 Text feature [doses] present in test data point [True]
480 Text feature [mapk] present in test data point [True]
Out of the top  500  features  30 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0482 0.2032 0.0108 0.0446 0.071  0.0164 0.5932 0.0078 0.0046]]
Actual Class : 7
--------------------------------------------------
23 Text feature [constitutively] present in test data point [True]
29 Text feature [constitutive] present in test data point [True]
47 Text feature [activated] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
89 Text feature [inhibited] present in test data point [True]
93 Text feature [transforming] present in test data point [True]
108 Text feature [transform] present in test data point [True]
148 Text feature [receptors] present in test data point [True]
177 Text feature [downstream] present in test data point [True]
210 Text feature [isozyme] present in test data point [True]
217 Text feature [activating] present in test data point [True]
232 Text feature [exchange] present in test data point [True]
326 Text feature [murine] present in test data point [True]
333 Text feature [agar] present in test data point [True]
334 Text feature [activation] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

4.3.2. Without Class balancing

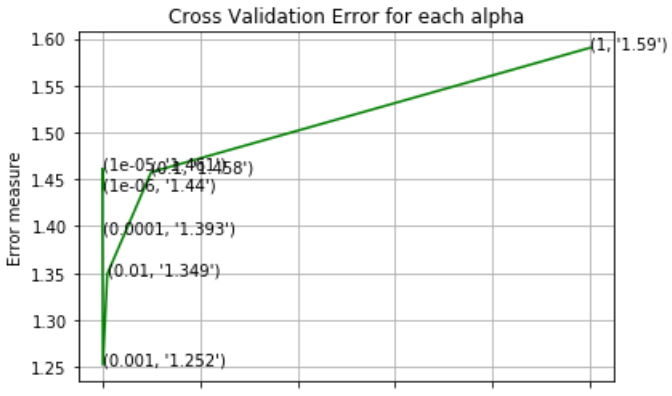
4.3.2.1. Hyper paramter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#-------------------------------



# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
```

```
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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_x_onehotCoding)
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_x_onehotCoding)
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 = 1e-06
Log Loss : 1.4395190222240433
for alpha = 1e-05
Log Loss : 1.4613951945118617
for alpha = 0.0001
Log Loss : 1.392640595913179
for alpha = 0.001
Log Loss : 1.2521811628755943
for alpha = 0.01
Log Loss : 1.349151219922669
for alpha = 0.1
Log Loss : 1.457591708320943
for alpha = 1
Log Loss : 1.5902258764770603
```



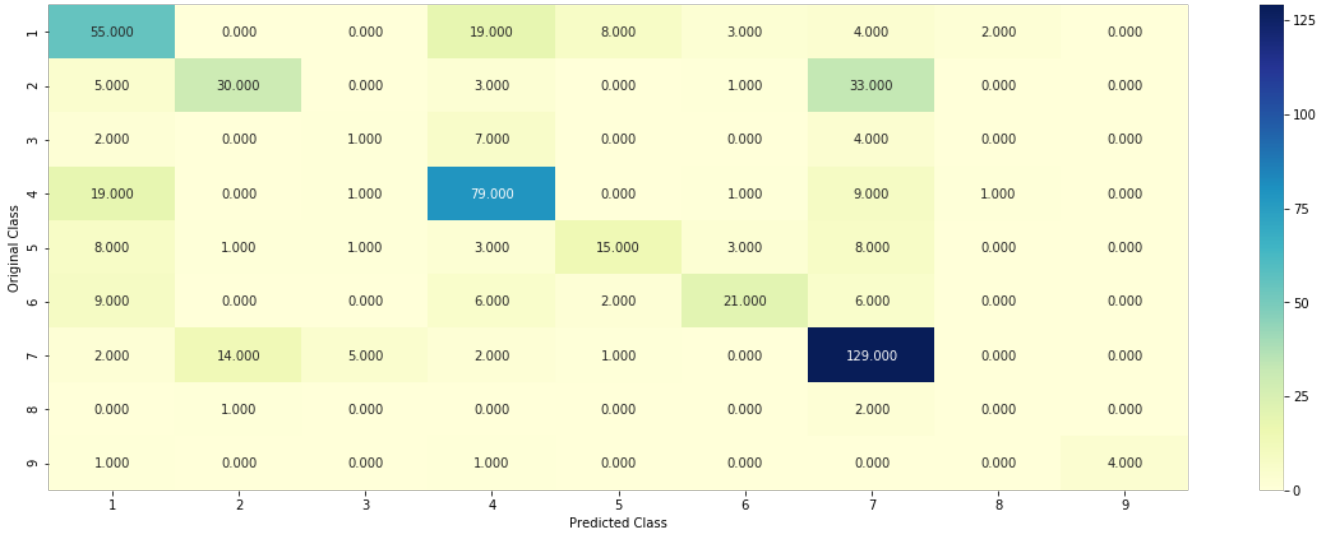

```
For values of best alpha =  0.001 The train log loss is: 0.6257422677412771
For values of best alpha =  0.001 The cross validation log loss is: 1.2521811628755943
For values of best alpha =  0.001 The test log loss is: 1.1306020069615057
```

4.3.2.2. Testing model with best hyper parameters

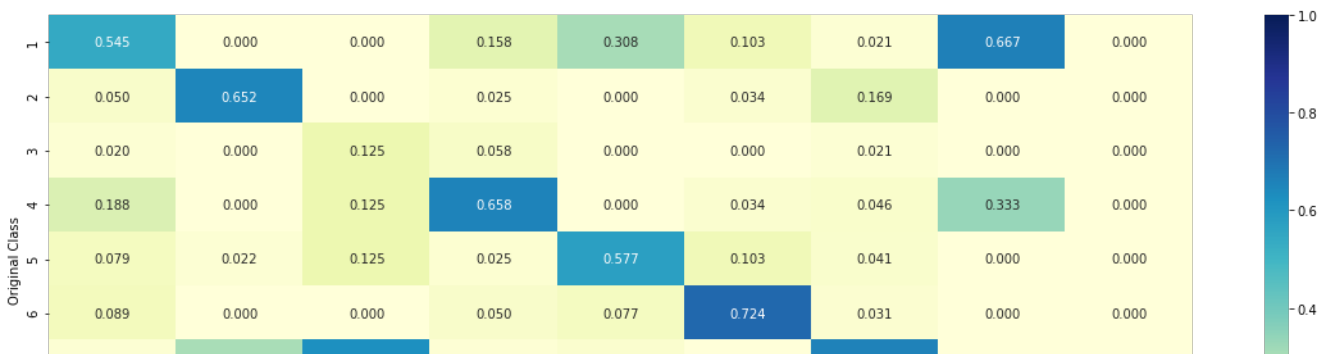
In [0]:

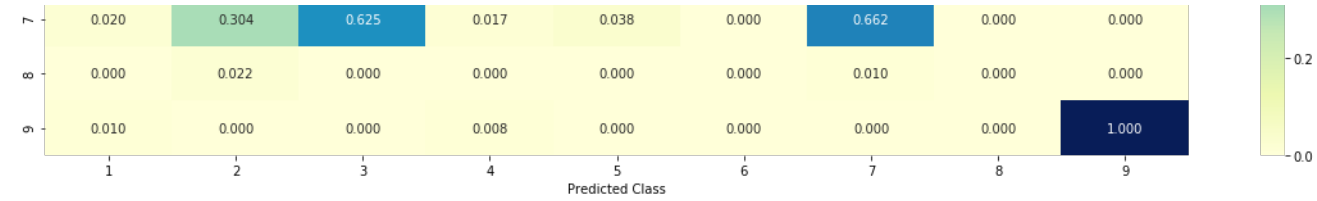
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link:
#-------------------------------

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.2521811628755943
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

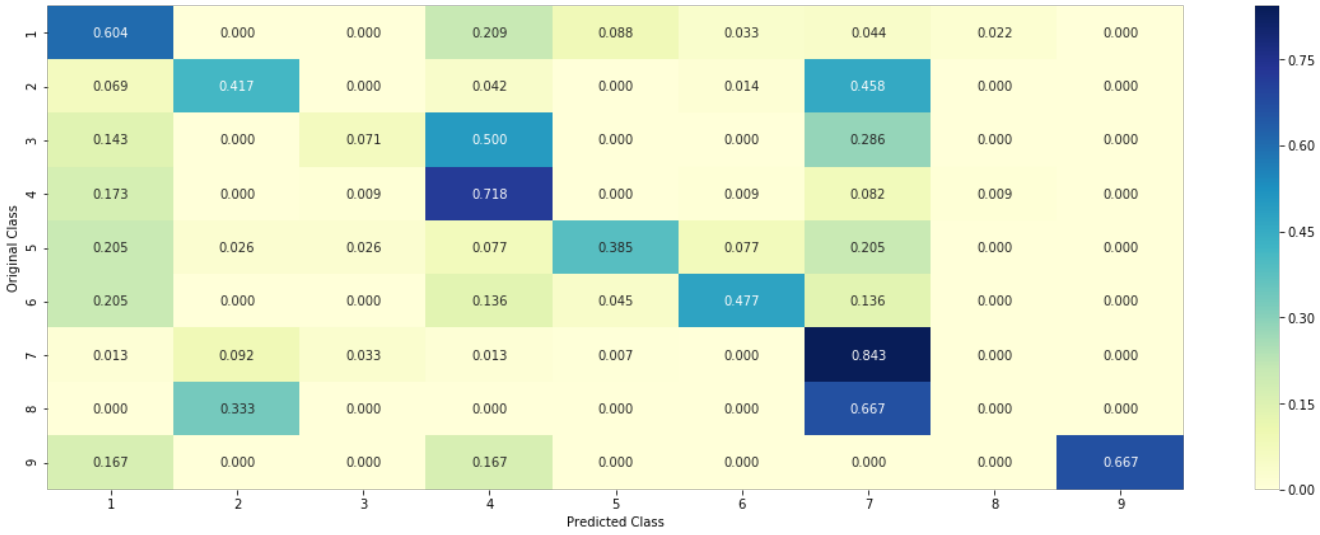



```
-------------------- Precision matrix (Column Sum=1) --------------------
```






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



4.3.2.3. Feature Importance, Correctly Classified point

In [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[5.100e-03 1.255e-01 2.000e-04 1.300e-03 2.300e-03 1.400e-03 8.556
e-01
  8.500e-03 1.000e-04]]
Actual Class : 7
--------------------------------------------------
60 Text feature [constitutively] present in test data point [True]
107 Text feature [flt1] present in test data point [True]
124 Text feature [cysteine] present in test data point [True]
157 Text feature [oncogenes] present in test data point [True]
158 Text feature [inhibited] present in test data point [True]
195 Text feature [activating] present in test data point [True]
200 Text feature [ligand] present in test data point [True]
203 Text feature [oncogene] present in test data point [True]
204 Text feature [technology] present in test data point [True]
257 Text feature [gaiix] present in test data point [True]
260 Text feature [concentrations] present in test data point [True]
265 Text feature [downstream] present in test data point [True]
314 Text feature [hki] present in test data point [True]
316 Text feature [dramatic] present in test data point [True]
323 Text feature [expressing] present in test data point [True]
371 Text feature [cdnas] present in test data point [True]
380 Text feature [viability] present in test data point [True]
```

```
412 Text feature [thyroid] present in test data point [True]
459 Text feature [activation] present in test data point [True]
461 Text feature [manageable] present in test data point [True]
462 Text feature [ser473] present in test data point [True]
468 Text feature [axilla] present in test data point [True]
495 Text feature [extracellular] present in test data point [True]
Out of the top  500  features  23 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0485 0.1851 0.0052 0.0442 0.0617 0.0143 0.6317 0.0072 0.0022]]
Actual Class : 7
--------------------------------------------------
60 Text feature [constitutively] present in test data point [True]
89 Text feature [constitutive] present in test data point [True]
116 Text feature [activated] present in test data point [True]
158 Text feature [inhibited] present in test data point [True]
159 Text feature [transforming] present in test data point [True]
193 Text feature [receptors] present in test data point [True]
195 Text feature [activating] present in test data point [True]
203 Text feature [oncogene] present in test data point [True]
226 Text feature [transform] present in test data point [True]
241 Text feature [isozyme] present in test data point [True]
265 Text feature [downstream] present in test data point [True]
377 Text feature [agar] present in test data point [True]
442 Text feature [interatomic] present in test data point [True]
459 Text feature [activation] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

4.4. Linear Support Vector Machines

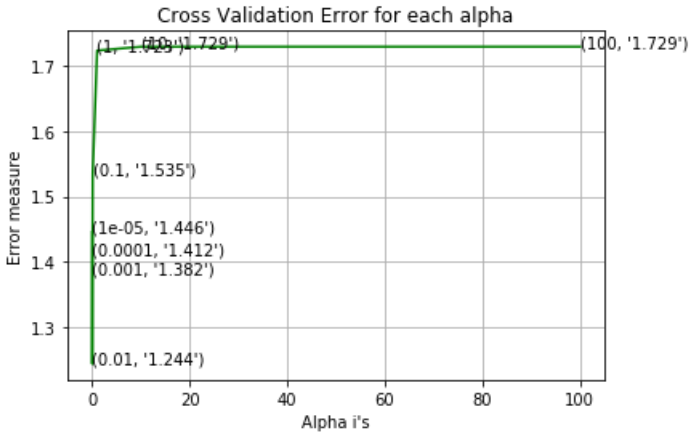
4.4.1. Hyper paramter tuning

In [0]:

```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# ---------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# ---------------------------------
```

```
# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state
=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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 = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', r
andom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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_x_onehotCoding)
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_x_onehotCoding)
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 C = 1e-05
Log Loss : 1.4456349250609233
for C = 0.0001
Log Loss : 1.4117883301099556
for C = 0.001
Log Loss : 1.3818342037841624
for C = 0.01
Log Loss : 1.2442964974823838
for C = 0.1
Log Loss : 1.5346828298587332
for C = 1
Log Loss : 1.722800653929441
for C = 10
Log Loss : 1.7286360420759161
for C = 100
Log Loss : 1.7286184454094997
```




```
For values of best alpha =  0.01 The train log loss is: 0.7628309867716067
For values of best alpha =  0.01 The cross validation log loss is: 1.2442964974823838
For values of best alpha =  0.01 The test log loss is: 1.1541891969863685
```

4.4.2. Testing model with best hyper parameters

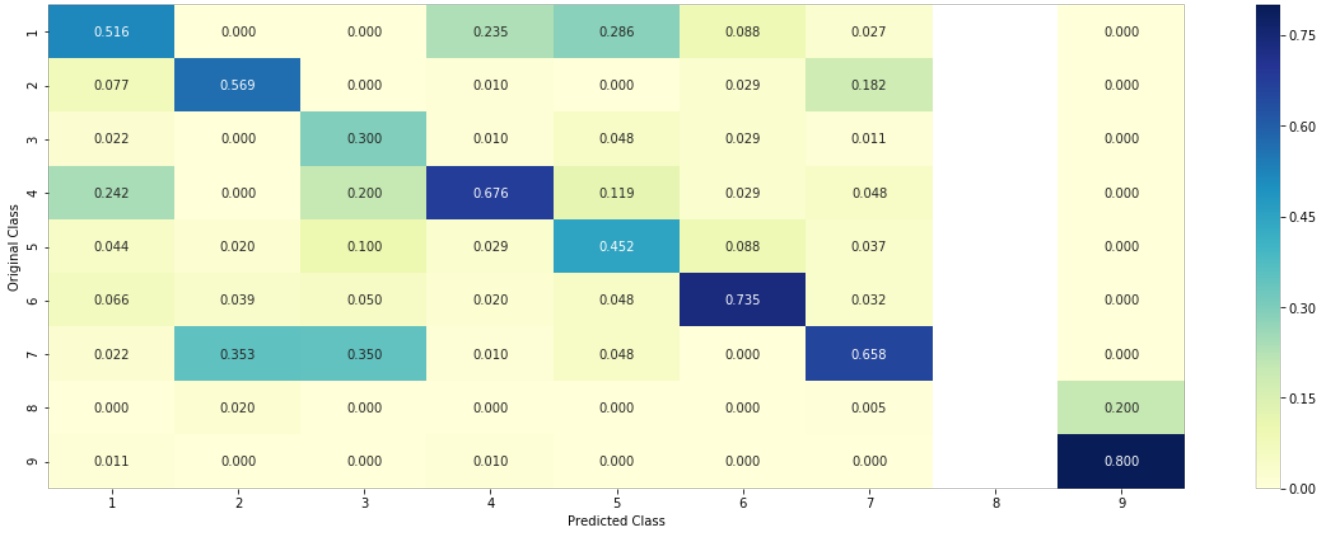
In [0]:

```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

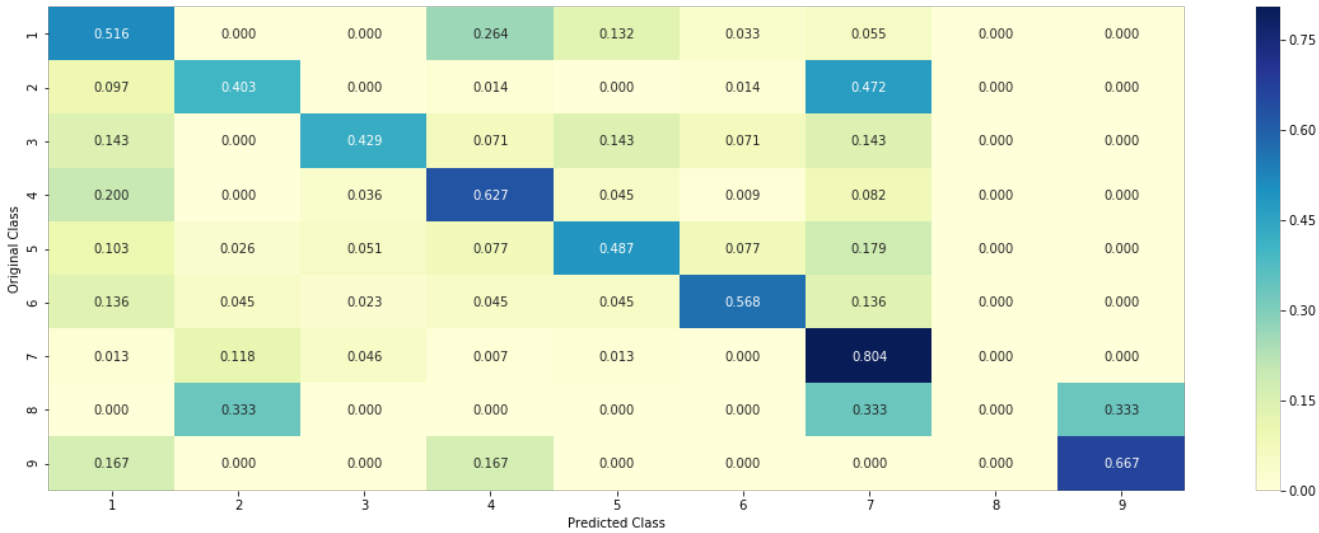
```
Log loss : 1.2442964974823838
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```

Predicted Class

---------- Precision matrix (Columm Sum=1) ----------



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

Predicted Class : 7

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0153 0.1199 0.0029 0.0151 0.0121 0.0075 0.8104 0.0129 0.0039]]
Actual Class : 7
--------------------------------------------------
28 Text feature [constitutively] present in test data point [True]
29 Text feature [cysteine] present in test data point [True]
49 Text feature [cdnas] present in test data point [True]
76 Text feature [flt1] present in test data point [True]
79 Text feature [concentrations] present in test data point [True]
82 Text feature [gaiix] present in test data point [True]
96 Text feature [technology] present in test data point [True]
101 Text feature [inhibited] present in test data point [True]
104 Text feature [activating] present in test data point [True]
114 Text feature [oncogenes] present in test data point [True]
147 Text feature [expressing] present in test data point [True]
150 Text feature [mapk] present in test data point [True]
151 Text feature [oncogene] present in test data point [True]
169 Text feature [thyroid] present in test data point [True]
171 Text feature [inhibitor] present in test data point [True]
205 Text feature [transduced] present in test data point [True]
211 Text feature [seeded] present in test data point [True]
230 Text feature [ligand] present in test data point [True]
255 Text feature [activation] present in test data point [True]
279 Text feature [downstream] present in test data point [True]
314 Text feature [doses] present in test data point [True]
351 Text feature [subcutaneous] present in test data point [True]
366 Text feature [atcc] present in test data point [True]
405 Text feature [melanocyte] present in test data point [True]
436 Text feature [hours] present in test data point [True]
445 Text feature [selleck] present in test data point [True]
446 Text feature [dramatic] present in test data point [True]
454 Text feature [chemiluminescence] present in test data point [True]
487 Text feature [viability] present in test data point [True]
489 Text feature [ser473] present in test data point [True]
Out of the top  500  features  30 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [0]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0786 0.1516 0.0146 0.1064 0.1105 0.0323 0.4839 0.0128 0.0094]]
Actual Class : 7
--------------------------------------------------
28 Text feature [constitutively] present in test data point [True]
40 Text feature [constitutive] present in test data point [True]
73 Text feature [activated] present in test data point [True]
75 Text feature [transforming] present in test data point [True]
94 Text feature [receptors] present in test data point [True]
97 Text feature [exchange] present in test data point [True]
101 Text feature [inhibited] present in test data point [True]
104 Text feature [activating] present in test data point [True]
151 Text feature [oncogene] present in test data point [True]
231 Text feature [transform] present in test data point [True]
255 Text feature [activation] present in test data point [True]
279 Text feature [downstream] present in test data point [True]
440 Text feature [doubled] present in test data point [True]
470 Text feature [substituting] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [0]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[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)
```

```
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss
is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss
is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2572535683354957
for n_estimators = 100 and max depth =  10
Log Loss : 1.1868414223711878
for n_estimators = 200 and max depth =  5
Log Loss : 1.2378734502517341
for n_estimators = 200 and max depth =  10
Log Loss : 1.1811031780258958
for n_estimators = 500 and max depth =  5
Log Loss : 1.2368241894319212
for n_estimators = 500 and max depth =  10
Log Loss : 1.176754594516683
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2357829533963691
for n_estimators = 1000 and max depth =  10
Log Loss : 1.174993079576866
for n_estimators = 2000 and max depth =  5
Log Loss : 1.236042392554891
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1759745074379755
For values of best estimator =  1000 The train log loss is: 0.7095396732082752
For values of best estimator =  1000 The cross validation log loss is: 1.174993079576866
For values of best estimator =  1000 The test log loss is: 1.1630923149103904
```

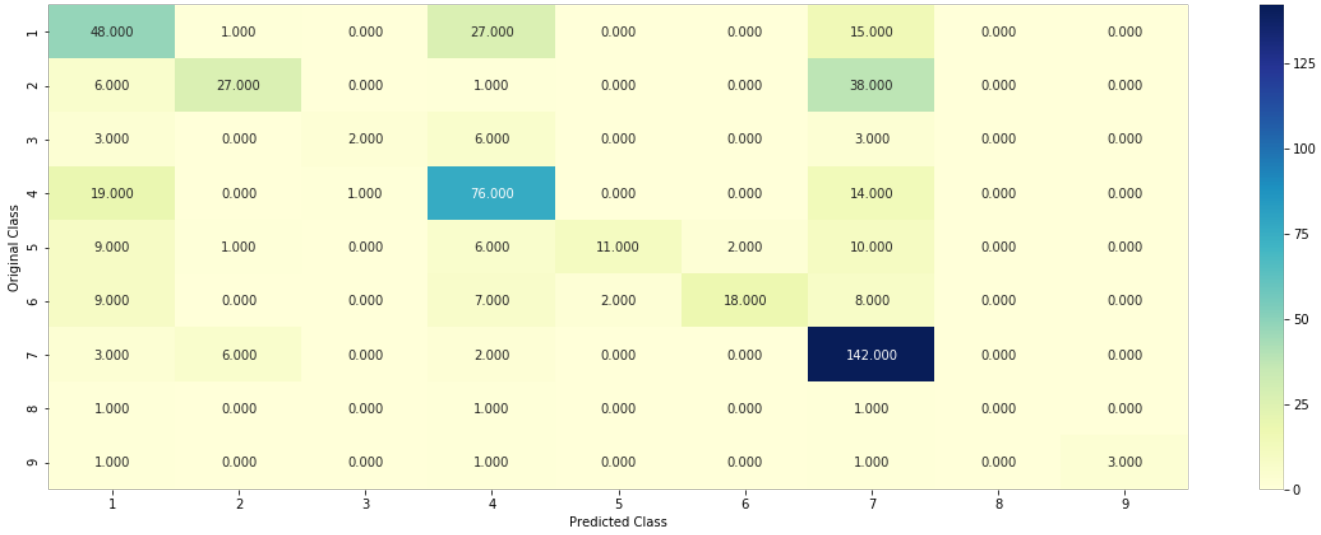
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [0]:

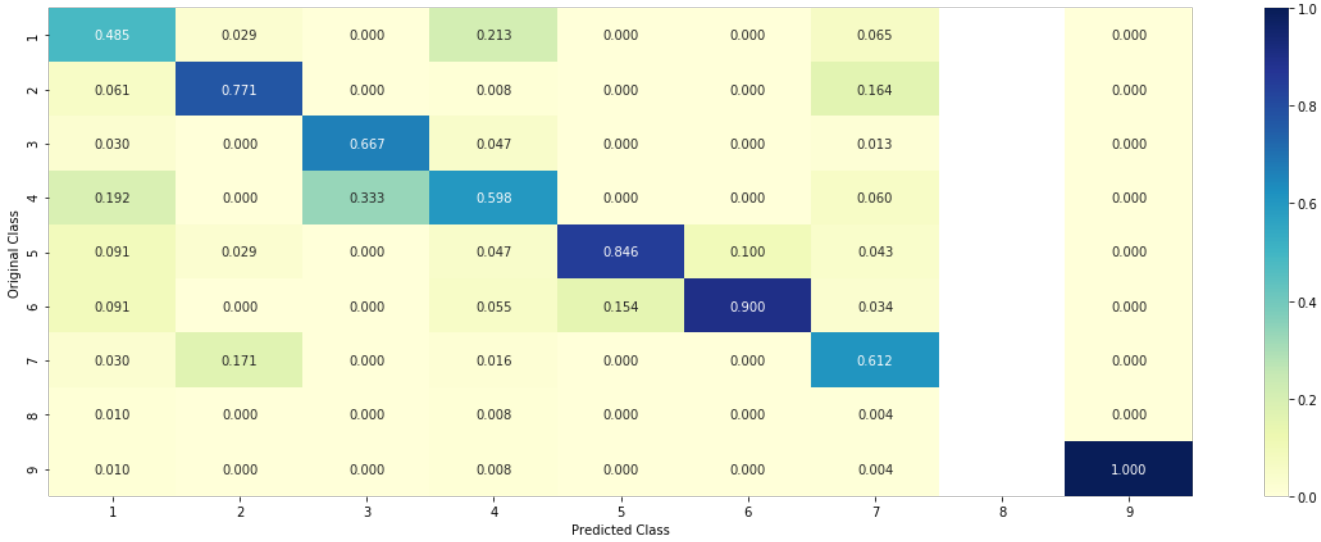
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.174993079576866
```

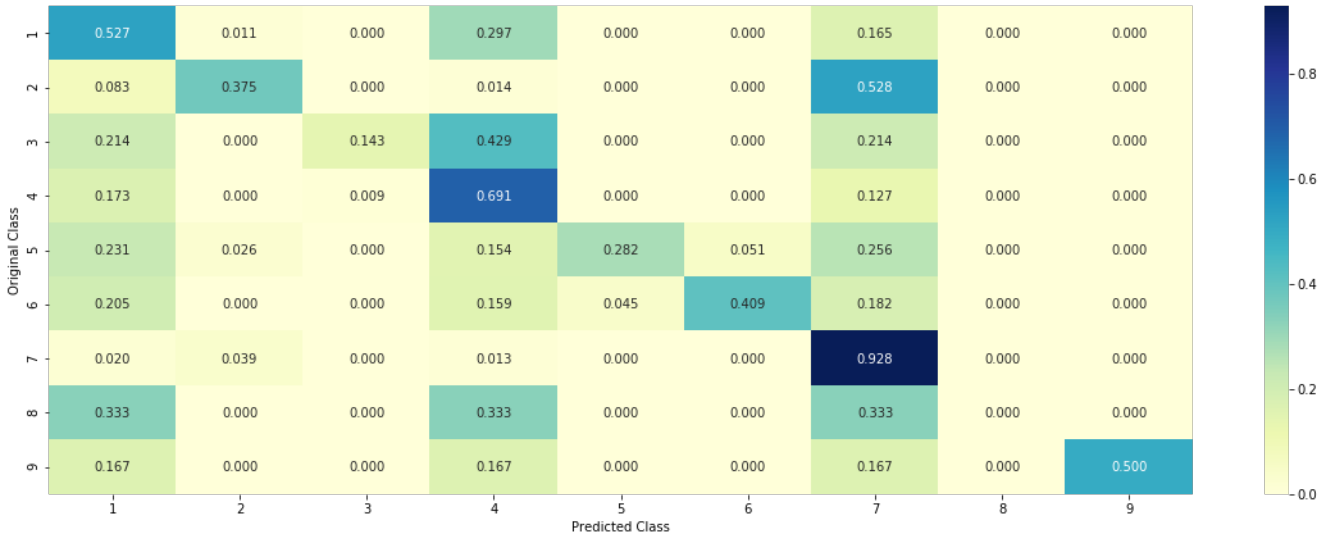
```
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [0]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0454 0.1404 0.0133 0.029  0.036  0.0294 0.6977 0.005  0.004 ]]
Actual Class : 7
--------------------------------------------------
0 Text feature [inhibitors] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [missense] present in test data point [True]
5 Text feature [inhibitor] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
9 Text feature [suppressor] present in test data point [True]
10 Text feature [activation] present in test data point [True]
11 Text feature [phosphorylation] present in test data point [True]
12 Text feature [kinases] present in test data point [True]
13 Text feature [nonsense] present in test data point [True]
14 Text feature [akt] present in test data point [True]
15 Text feature [function] present in test data point [True]
17 Text feature [erk] present in test data point [True]
19 Text feature [growth] present in test data point [True]
20 Text feature [variants] present in test data point [True]
22 Text feature [frameshift] present in test data point [True]
24 Text feature [therapeutic] present in test data point [True]
25 Text feature [functional] present in test data point [True]
28 Text feature [signaling] present in test data point [True]
30 Text feature [patients] present in test data point [True]
31 Text feature [cells] present in test data point [True]
32 Text feature [constitutively] present in test data point [True]
34 Text feature [trials] present in test data point [True]
35 Text feature [therapy] present in test data point [True]
37 Text feature [erk1] present in test data point [True]
38 Text feature [activate] present in test data point [True]
39 Text feature [downstream] present in test data point [True]
41 Text feature [efficacy] present in test data point [True]
42 Text feature [protein] present in test data point [True]
43 Text feature [loss] present in test data point [True]
44 Text feature [inhibited] present in test data point [True]
45 Text feature [expressing] present in test data point [True]
46 Text feature [pten] present in test data point [True]
48 Text feature [lines] present in test data point [True]
49 Text feature [treated] present in test data point [True]
50 Text feature [proliferation] present in test data point [True]
51 Text feature [drug] present in test data point [True]
57 Text feature [mek] present in test data point [True]
59 Text feature [inhibition] present in test data point [True]
61 Text feature [repair] present in test data point [True]
62 Text feature [sensitivity] present in test data point [True]
64 Text feature [receptor] present in test data point [True]
66 Text feature [assays] present in test data point [True]
68 Text feature [survival] present in test data point [True]
```

```
69 Text feature [cell] present in test data point [True]
71 Text feature [ligand] present in test data point [True]
73 Text feature [expression] present in test data point [True]
74 Text feature [variant] present in test data point [True]
75 Text feature [oncogene] present in test data point [True]
78 Text feature [extracellular] present in test data point [True]
79 Text feature [doses] present in test data point [True]
80 Text feature [mapk] present in test data point [True]
81 Text feature [hours] present in test data point [True]
84 Text feature [information] present in test data point [True]
86 Text feature [harboring] present in test data point [True]
90 Text feature [dna] present in test data point [True]
91 Text feature [concentrations] present in test data point [True]
92 Text feature [likelihood] present in test data point [True]
93 Text feature [months] present in test data point [True]
94 Text feature [binding] present in test data point [True]
96 Text feature [imatinib] present in test data point [True]
98 Text feature [preclinical] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1337 0.116  0.0224 0.1773 0.0674 0.0545 0.4156 0.0071 0.0059]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [inhibitors] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
6 Text feature [activated] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
10 Text feature [activation] present in test data point [True]
11 Text feature [phosphorylation] present in test data point [True]
12 Text feature [kinases] present in test data point [True]
14 Text feature [akt] present in test data point [True]
15 Text feature [function] present in test data point [True]
19 Text feature [growth] present in test data point [True]
21 Text feature [constitutive] present in test data point [True]
25 Text feature [functional] present in test data point [True]
28 Text feature [signaling] present in test data point [True]
31 Text feature [cells] present in test data point [True]
32 Text feature [constitutively] present in test data point [True]
38 Text feature [activate] present in test data point [True]
39 Text feature [downstream] present in test data point [True]
42 Text feature [protein] present in test data point [True]
43 Text feature [loss] present in test data point [True]
44 Text feature [inhibited] present in test data point [True]
46 Text feature [pten] present in test data point [True]
47 Text feature [transforming] present in test data point [True]
48 Text feature [lines] present in test data point [True]
50 Text feature [proliferation] present in test data point [True]
53 Text feature [neutral] present in test data point [True]
55 Text feature [transform] present in test data point [True]
56 Text feature [stability] present in test data point [True]
58 Text feature [transformation] present in test data point [True]
59 Text feature [inhibition] present in test data point [True]
62 Text feature [sensitivity] present in test data point [True]
64 Text feature [receptor] present in test data point [True]
66 Text feature [assays] present in test data point [True]
```

```
69 Text feature [cell] present in test data point [True]
75 Text feature [oncogene] present in test data point [True]
84 Text feature [information] present in test data point [True]
90 Text feature [dna] present in test data point [True]
94 Text feature [binding] present in test data point [True]
Out of the top  100  features  39 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [0]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[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 = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max
_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:"
,log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_
test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2657048897349608
for n_estimators = 10 and max depth =  3
Log Loss : 1.7459205010556096
for n_estimators = 10 and max depth =  5
Log Loss : 1.4368353925512503
for n_estimators = 10 and max depth =  10
Log Loss : 1.904597809032912
for n_estimators = 50 and max depth =  2
Log Loss : 1.7221951095007484
for n_estimators = 50 and max depth =  3
Log Loss : 1.4984825877845531
for n_estimators = 50 and max depth =  5
Log Loss : 1.4593628982873716
for n_estimators = 50 and max depth =  10
Log Loss : 1.8434939703555409
for n_estimators = 100 and max depth =  2
Log Loss : 1.6182209245331227
for n_estimators = 100 and max depth =  3
Log Loss : 1.5199297988828253
for n_estimators = 100 and max depth =  5
Log Loss : 1.4177501184246677
for n_estimators = 100 and max depth =  10
Log Loss : 1.8227504417195126
for n_estimators = 200 and max depth =  2
Log Loss : 1.6622571648074496
for n_estimators = 200 and max depth =  3
Log Loss : 1.4800771339141767
for n_estimators = 200 and max depth =  5
Log Loss : 1.4412060242341358
for n_estimators = 200 and max depth =  10
Log Loss : 1.7892406351442258
for n_estimators = 500 and max depth =  2
Log Loss : 1.715950314170445
for n_estimators = 500 and max depth =  3
Log Loss : 1.5658682738699774
for n_estimators = 500 and max depth =  5
Log Loss : 1.4445360301518217
for n_estimators = 500 and max depth =  10
Log Loss : 1.8421097596928397
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6834927870864949
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5631973035931377
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4449980792724129
for n_estimators = 1000 and max depth =  10
Log Loss : 1.85233132619749
For values of best alpha =  100 The train log loss is: 0.060702709444608406
For values of best alpha =  100 The cross validation log loss is: 1.417750118424668
For values of best alpha =  100 The test log loss is: 1.3806278998341923
```

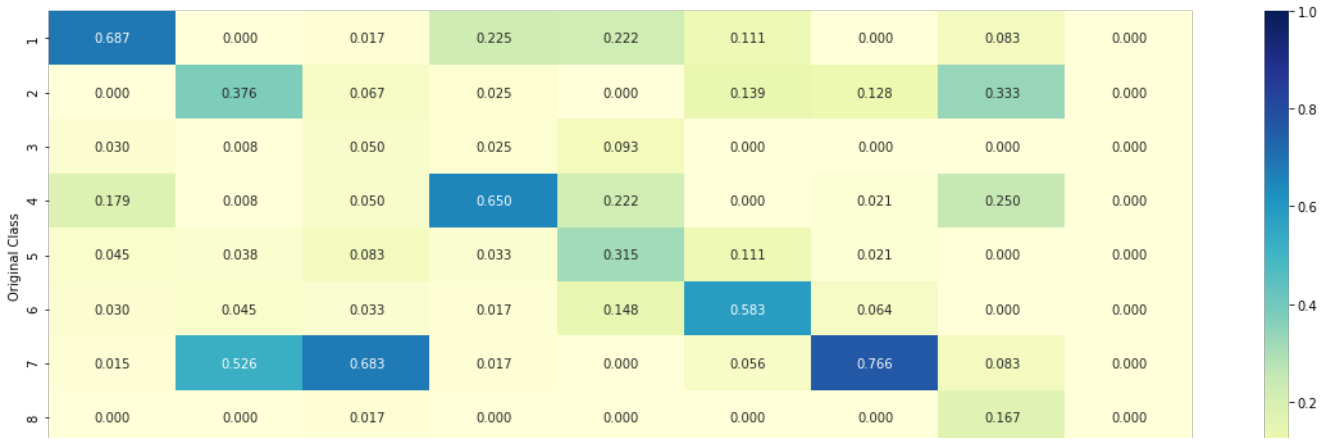
4.5.4. Testing model with best hyper parameters (Response Coding)

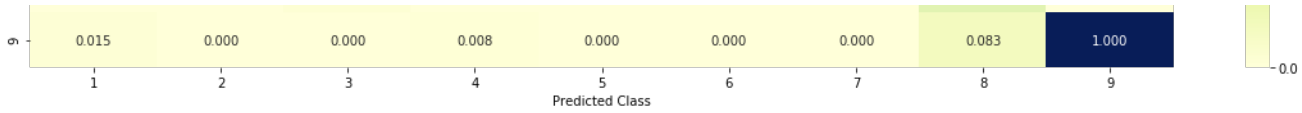
In [0]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# ------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)],
n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.4177501184246677
Number of mis-classified points : 0.518796992481203
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [0]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max
_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0143 0.5044 0.1471 0.0191 0.0245 0.065  0.1724 0.039  0.0142]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
```

```
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [0]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0281 0.2006 0.203  0.0857 0.0626 0.0906 0.2249 0.0676 0.0369]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#-------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# -------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

# -------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# -------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0
)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
```

```
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

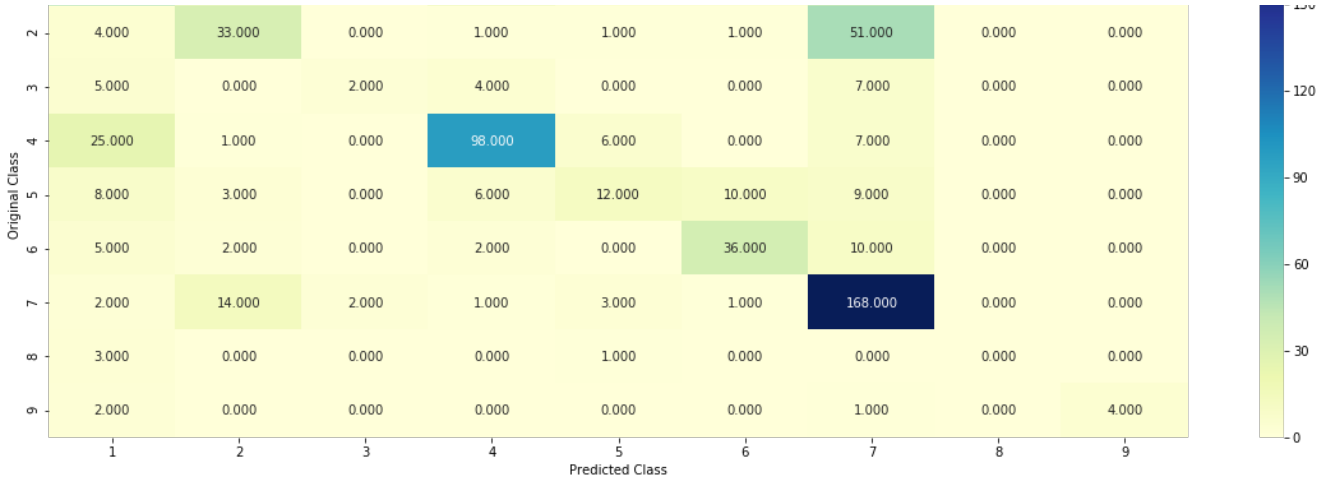
```
Logistic Regression :  Log Loss: 1.24
Support vector machines : Log Loss: 1.72
Naive Bayes : Log Loss: 1.37
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.049
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.577
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.224
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.366
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.690
```

4.7.2 testing the model with the best hyper parameters

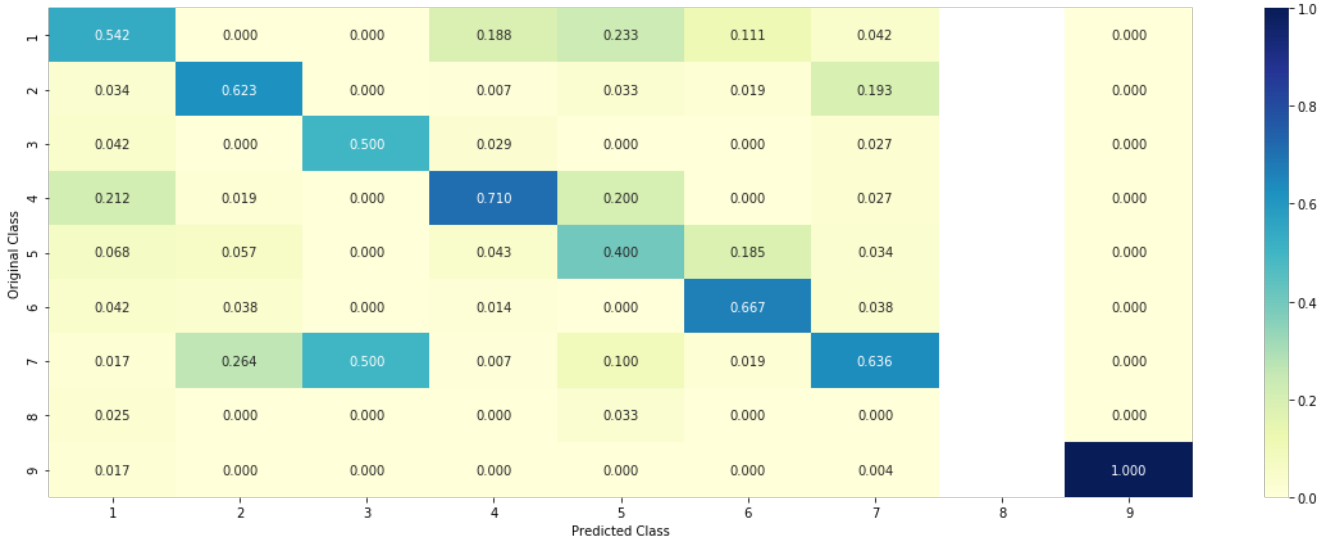
In [0]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_onehotCoding)-
test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

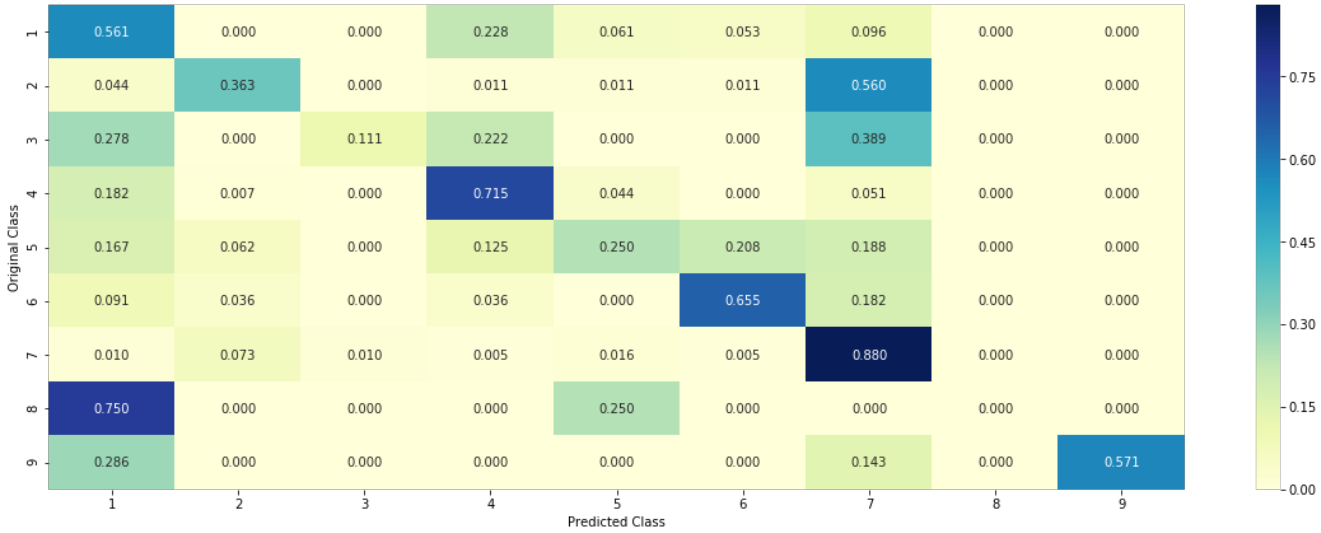
```
Log loss (train) on the stacking classifier : 0.6760284396805781
Log loss (CV) on the stacking classifier : 1.2243084610674686
Log loss (test) on the stacking classifier : 1.1562525475350196
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```



---------- Precision matrix (Columm Sum=1) ----------



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



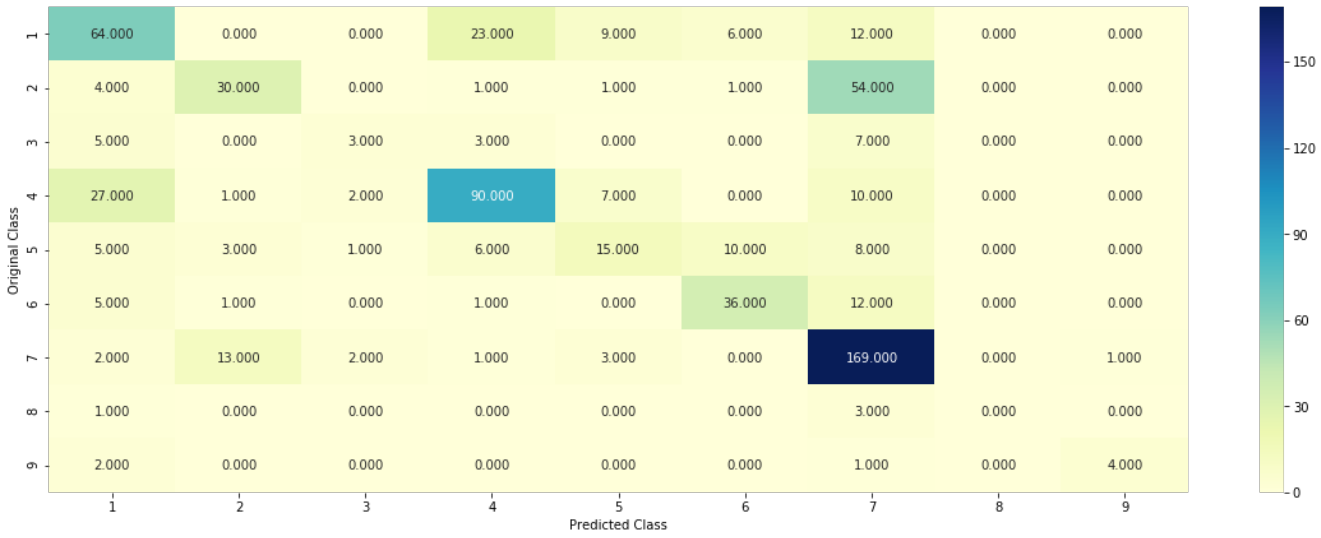
4.7.3 Maximum Voting classifier

In [0]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
```

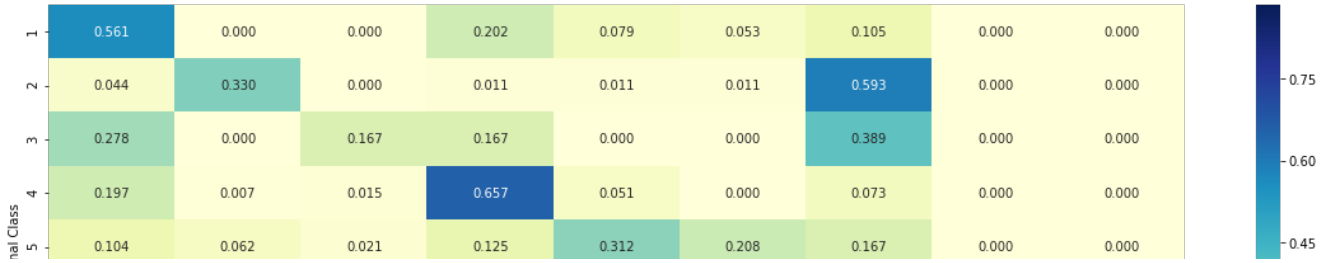
```
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y,
vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)-
test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

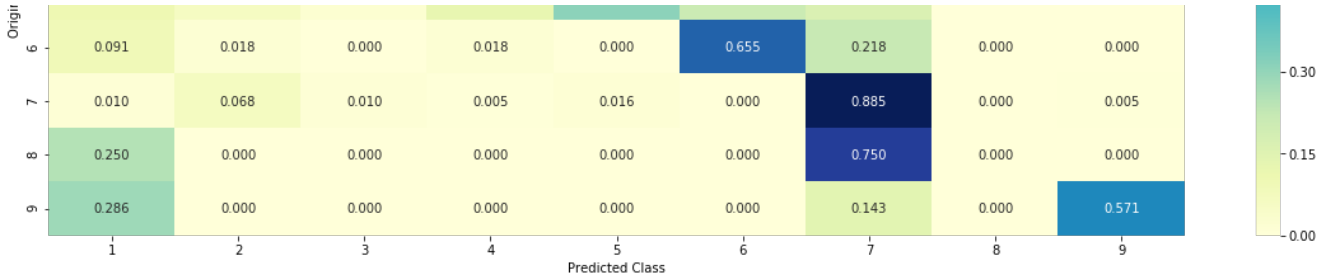
```
Log loss (train) on the VotingClassifier : 0.9407598679043604
Log loss (CV) on the VotingClassifier : 1.2835402100341697
Log loss (test) on the VotingClassifier : 1.223278167176945
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```


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





5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

Before applying TFIDF vectorizer to Text feature, log loss for all the models is:

In [25]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model","Vectorizer","Train log loss","Test log loss","Misclassified"]

x.add_row(["Naive Bayes","One Hot coding",0.903,1.35,44.36])
x.add_row(["KNN","Response coding",0.705,1.121,39.66])
x.add_row(["Logistic Regression with class balancing","One Hot coding",0.615,1.217,38.34])
x.add_row(["Logistic Regression without class balancing","One Hot coding",0.625,1.252,37.21])
x.add_row(["Linear SVM","One Hot coding",0.762,1.244,39.47])
x.add_row(["Random Forest","One Hot coding",0.709,1.174,38.53])
x.add_row(["Random Forest","Response coding",0.607,1.417,51.87])
x.add_row(["Stacking (logistic + Linear + NB)","One Hot coding",0.676,1.156,37.29])

print(x)
```

```
+-----------------------------------------+----------------+---------------+---------------+-
-----------+
|                  Model                  |    Vectorizer  | Train log loss | Test log loss |
isclassified |
+-----------------------------------------+----------------+---------------+---------------+-
-----------+
|              Naive Bayes                |  One Hot coding |     0.903      |      1.35     |
44.36     |
|                  KNN                    | Response coding |     0.705      |     1.121     |
39.66     |
|   Logistic Regression with class balancing  |  One Hot coding |     0.615      |     1.217     |
38.34     |
| Logistic Regression without class balancing |  One Hot coding |     0.625      |     1.252     |
37.21     |
|               Linear SVM                |  One Hot coding |     0.762      |     1.244     |
39.47     |
|              Random Forest              |  One Hot coding |     0.709      |     1.174     |
38.53     |
|              Random Forest              | Response coding |     0.607      |     1.417     |
51.87     |
|      Stacking (logistic + Linear + NB)      |  One Hot coding |     0.676      |     1.156     |
37.29     |
+-----------------------------------------+----------------+---------------+---------------+-
-----------+
```

Assignment starting with Text TFIDF vectorizztion

In [16]:

In [16]:

```
# building a TFIDF features with top 1k values

from sklearn.feature_extraction.text import TfidfVectorizer
text_TFIDF_vectorizer = TfidfVectorizer(max_features=1000)
train_text_feature_TFIDF = text_TFIDF_vectorizer.fit_transform(train_df['TEXT'])

print(train_text_feature_TFIDF.shape)
```

```
(2124, 1000)
```

In [17]:

```
# don't forget to normalize every feature
train_text_feature_TFIDF = normalize(train_text_feature_TFIDF, axis=0)
# we use the same vectorizer that was trained on train data
test_text_feature_TFIDF = text_TFIDF_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_TFIDF = normalize(test_text_feature_TFIDF, axis=0)
# we use the same vectorizer that was trained on train data
cv_text_feature_TFIDF = text_TFIDF_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_TFIDF = normalize(cv_text_feature_TFIDF, axis=0)

print("After vectorizing using TFIDF: ")
print(train_text_feature_TFIDF.shape)
print(test_text_feature_TFIDF.shape)
print(cv_text_feature_TFIDF.shape)
```

```
After vectorizing using TFIDF:
(2124, 1000)
(665, 1000)
(532, 1000)
```

Stacking the three types of features (TFIDF)

In [35]:

```
train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)
)

train_x_TFIDF = hstack((train_gene_var_onehotCoding, train_text_feature_TFIDF)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_TFIDF = hstack((test_gene_var_onehotCoding, test_text_feature_TFIDF)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_TFIDF = hstack((cv_gene_var_onehotCoding, cv_text_feature_TFIDF)).tocsr()
cv_y = np.array(list(cv_df['Class']))

print("After stacking all 3 features using TFIDF on Text: ")
print(train_x_TFIDF.shape,train_y.shape)
print(test_x_TFIDF.shape,test_y.shape)
print(cv_x_TFIDF.shape,cv_y.shape)
```

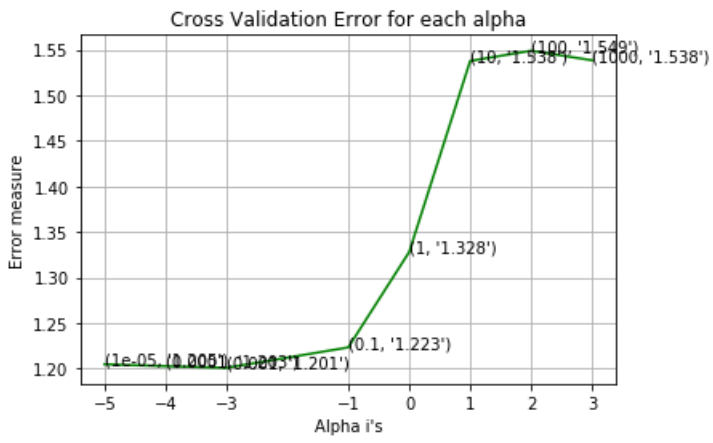
```
After stacking all 3 features using TFIDF on Text:
(2124, 3196) (2124,)
(665, 3196) (665,)
(532, 3196) (532,)
```

2.1 Naive Bayes Model

In [32]:

```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_TFIDF, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_TFIDF, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_TFIDF)
    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(np.log10(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)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
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 = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_TFIDF, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TFIDF, train_y)


predict_y = sig_clf.predict_proba(train_x_TFIDF)
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_x_TFIDF)
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_x_TFIDF)
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 = 1e-05
Log Loss : 1.2047334591779086
for alpha = 0.0001
Log Loss : 1.2026781080790947
for alpha = 0.001
Log Loss : 1.2007873226379804
for alpha = 0.1
Log Loss : 1.2232256413956302
for alpha = 1
Log Loss : 1.3279720353930082
for alpha = 10
Log Loss : 1.5376623375969418
for alpha = 100
Log Loss : 1.5488636859152092
for alpha = 1000
Log Loss : 1.5383763014913063
```



```
For values of best alpha =  0.001 The train log loss is: 0.5276155785830785
For values of best alpha =  0.001 The cross validation log loss is: 1.2007873226379804
For values of best alpha =  0.001 The test log loss is: 1.2335445927557736
```

In [33]:

```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_TFIDF, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TFIDF, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_TFIDF)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_TFIDF)- cv_y))/cv
_y.shape[0])
```

```
Log Loss : 1.2007873226379804
Number of missclassified point : 0.3609022556390977
```

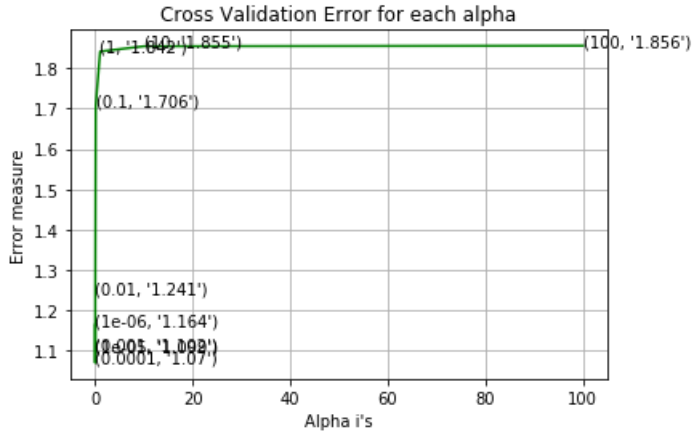
2.2 Logistic Regression with class balancing

In [35]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_TFIDF, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_TFIDF, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_TFIDF)
    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 = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_TFIDF, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TFIDF, train_y)

predict_y = sig_clf.predict_proba(train_x_TFIDF)
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_x_TFIDF)
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_x_TFIDF)
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 = 1e-06
Log Loss : 1.1640984872805014
for alpha = 1e-05
Log Loss : 1.0994283888493437
for alpha = 0.0001
Log Loss : 1.0698515780299018
```

```
for alpha = 0.001
Log Loss : 1.1017328786846927
for alpha = 0.01
Log Loss : 1.240964575159249
for alpha = 0.1
Log Loss : 1.706439570761755
for alpha = 1
Log Loss : 1.8416147646386647
for alpha = 10
Log Loss : 1.8545938014184382
for alpha = 100
Log Loss : 1.8558721533669684
```

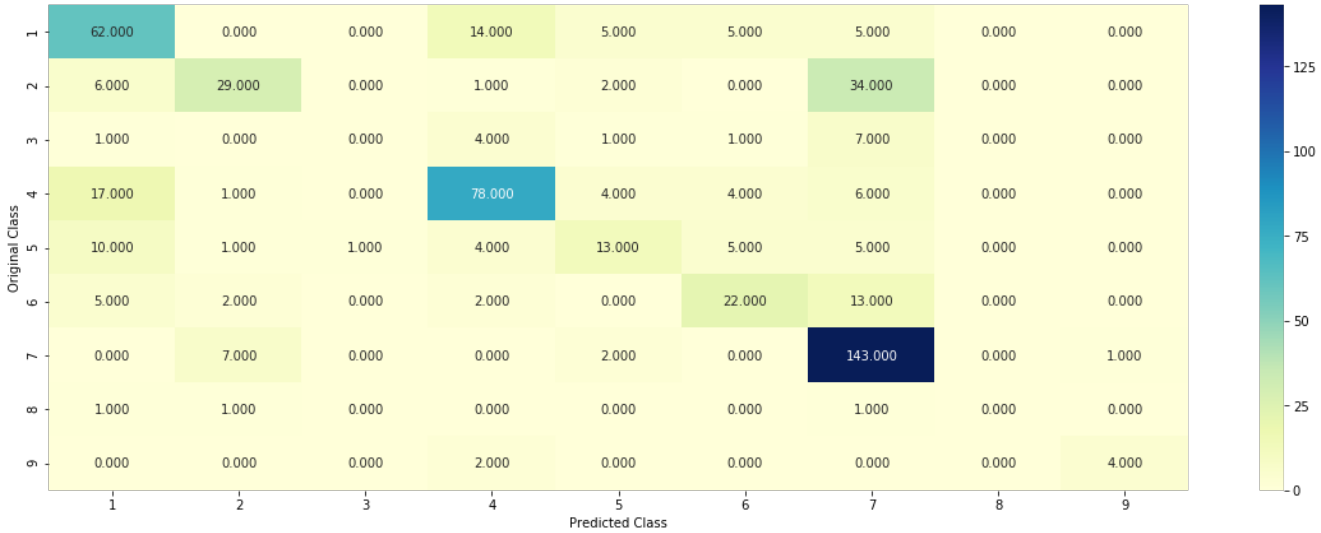



```
For values of best alpha =  0.0001 The train log loss is: 0.45137459889435366
For values of best alpha =  0.0001 The cross validation log loss is: 1.0698515780299018
For values of best alpha =  0.0001 The test log loss is: 1.0202008706431422
```

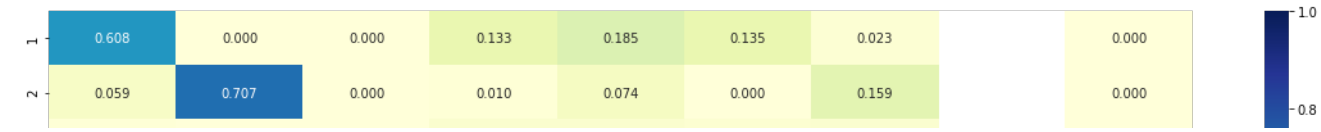
In [41]:

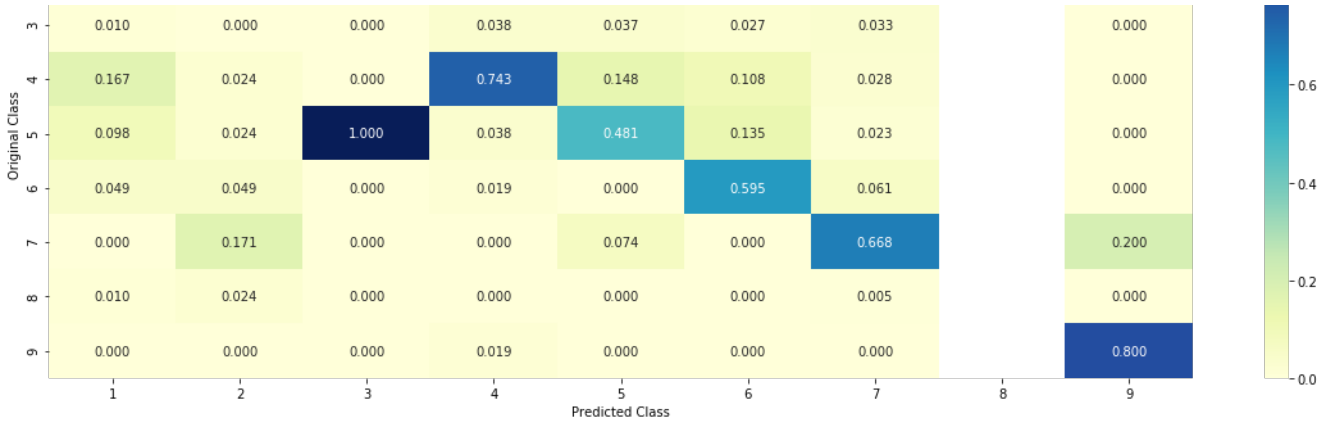
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_TFIDF, train_y, cv_x_TFIDF, cv_y, clf)
```

```
Log loss : 1.0698515780299018
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```

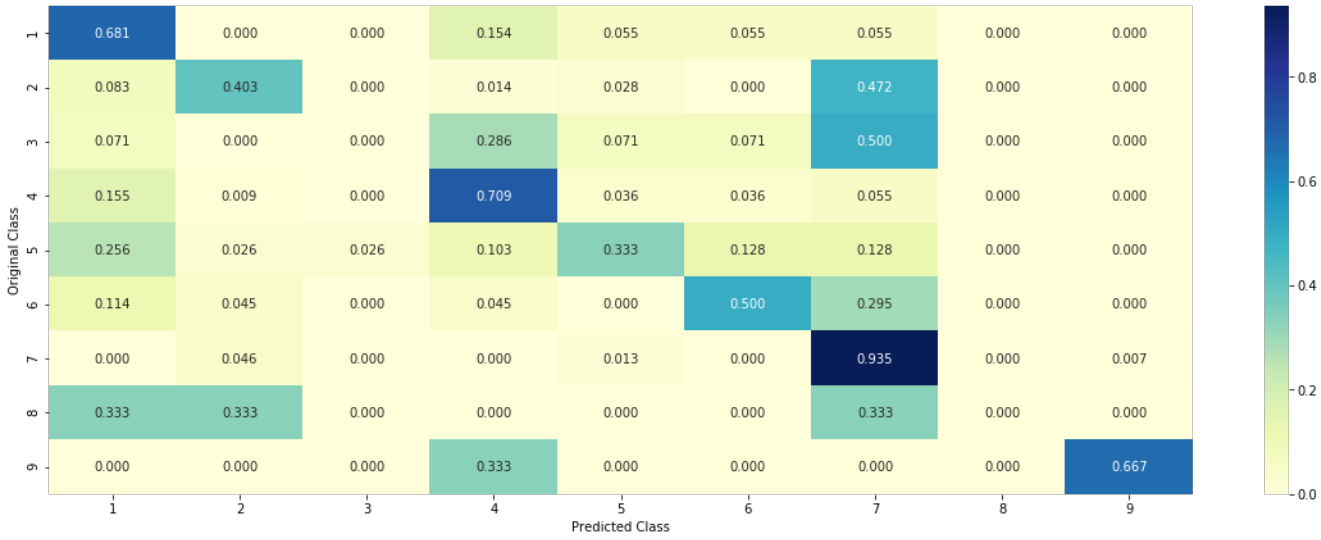



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



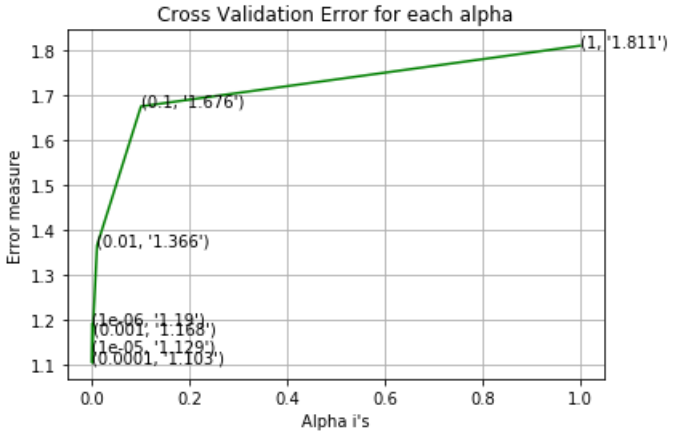
Logistic Regression without class balancing

In [42]:

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_TFIDF, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_TFIDF, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_TFIDF)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_TFIDF, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TFIDF, train_y)

predict_y = sig_clf.predict_proba(train_x_TFIDF)
```

```
predict_y = sig_clf.predict_proba(train_x_TFIDF)
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_x_TFIDF)
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_x_TFIDF)
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 = 1e-06
Log Loss : 1.1896380293190405
for alpha = 1e-05
Log Loss : 1.1294572545663173
for alpha = 0.0001
Log Loss : 1.103390089620745
for alpha = 0.001
Log Loss : 1.1678039496458352
for alpha = 0.01
Log Loss : 1.3658813187806296
for alpha = 0.1
Log Loss : 1.67599794287723
for alpha = 1
Log Loss : 1.8113226961634403
```

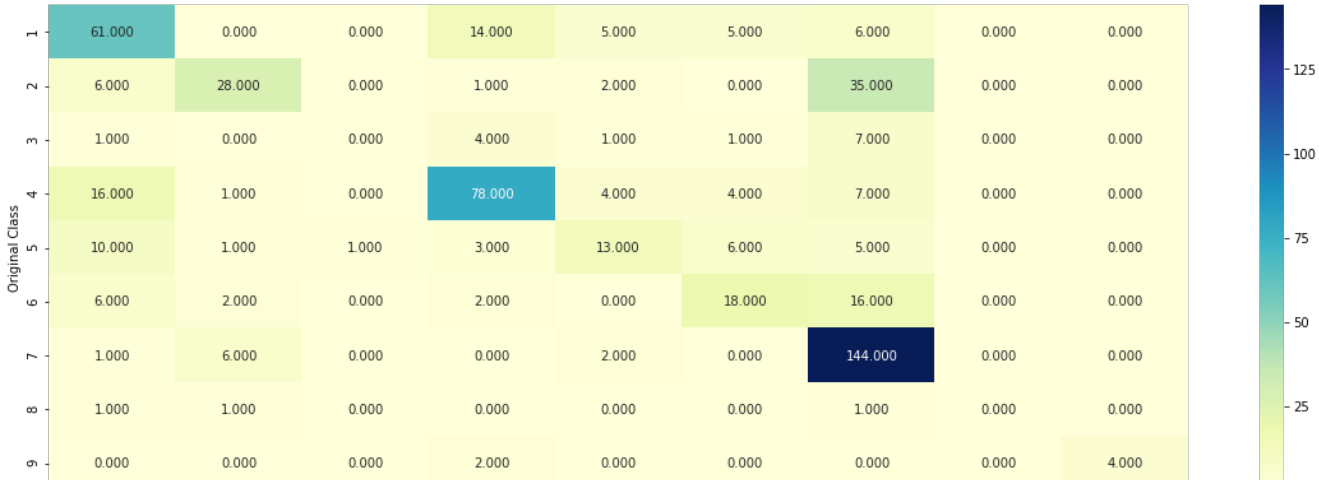


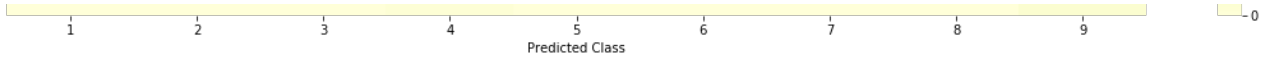

```
For values of best alpha =  0.0001 The train log loss is: 0.4470978370327045
For values of best alpha =  0.0001 The cross validation log loss is: 1.103390089620745
For values of best alpha =  0.0001 The test log loss is: 1.0477494238980793
```

In [43]:

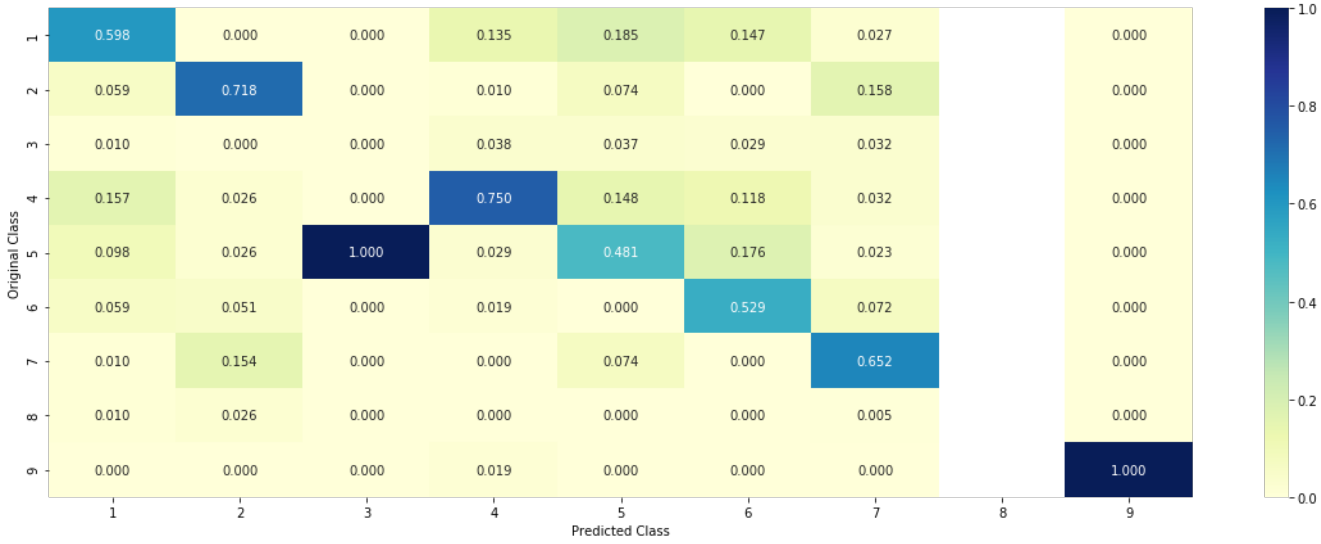
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_TFIDF, train_y, cv_x_TFIDF, cv_y, clf)
```

```
Log loss : 1.103390089620745
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```

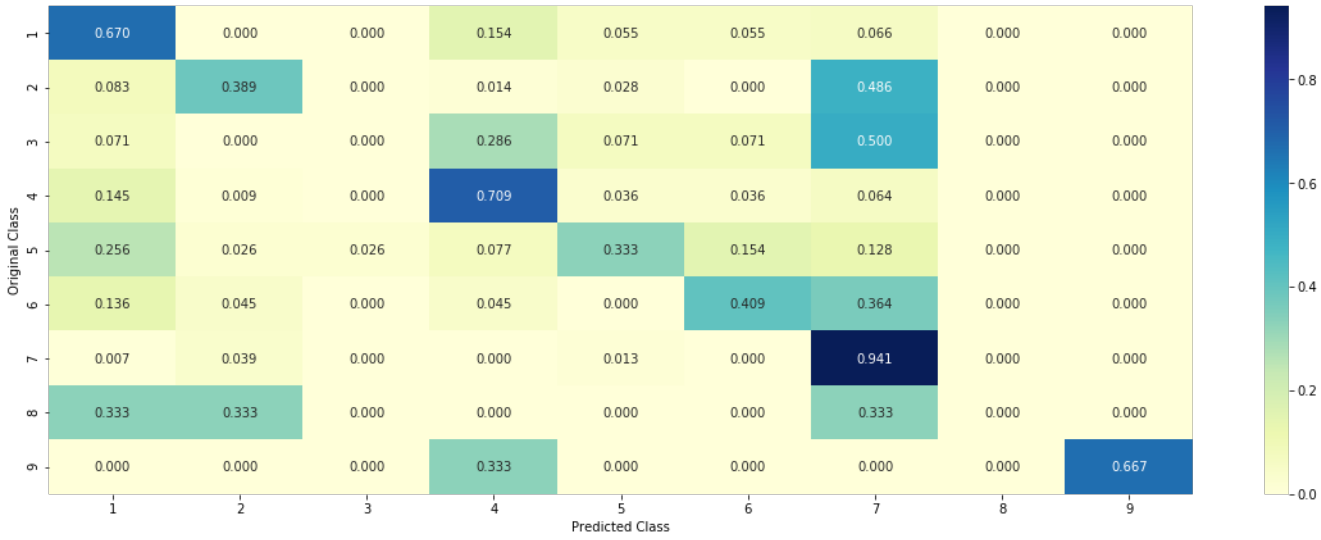




---------- Precision matrix (Columm Sum=1) ----------



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



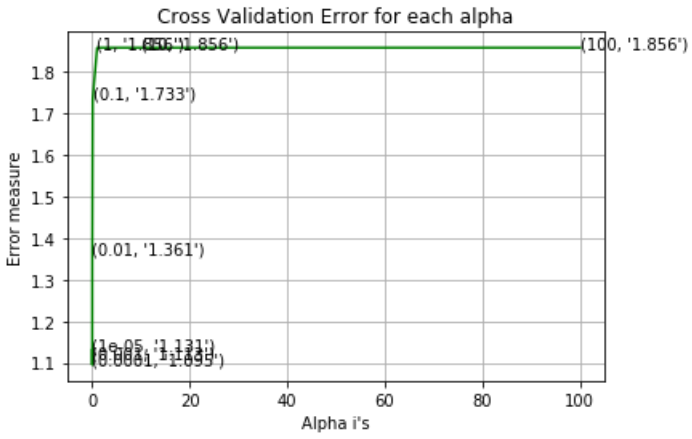
2.4 Linear SVM

In [44]:

```
alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state
=42)
    clf.fit(train_x_TFIDF, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_TFIDF, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_TFIDF)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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 = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', r
andom_state=42)
clf.fit(train_x_TFIDF, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TFIDF, train_y)

predict_y = sig_clf.predict_proba(train_x_TFIDF)
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_x_TFIDF)
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_x_TFIDF)
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 C = 1e-05
Log Loss : 1.130865145894693
for C = 0.0001
Log Loss : 1.0951223568610815
for C = 0.001
Log Loss : 1.1129659184560923
for C = 0.01
Log Loss : 1.3610216293271344
for C = 0.1
Log Loss : 1.7333179076001264
for C = 1
Log Loss : 1.855939598461702
for C = 10
Log Loss : 1.8559398858520153
for C = 100
Log Loss : 1.8559401667619306
```

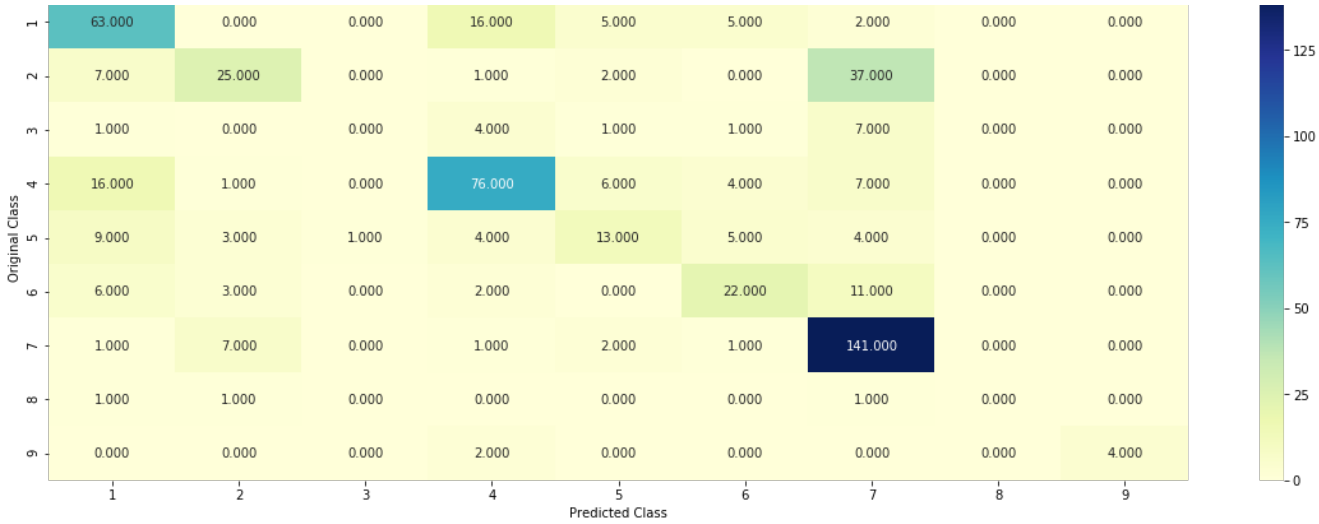



```
For values of best alpha =  0.0001 The train log loss is: 0.40536330659458014
For values of best alpha =  0.0001 The cross validation log loss is: 1.0951223568610815
For values of best alpha =  0.0001 The test log loss is: 1.0521796358899635
```

In [45]:

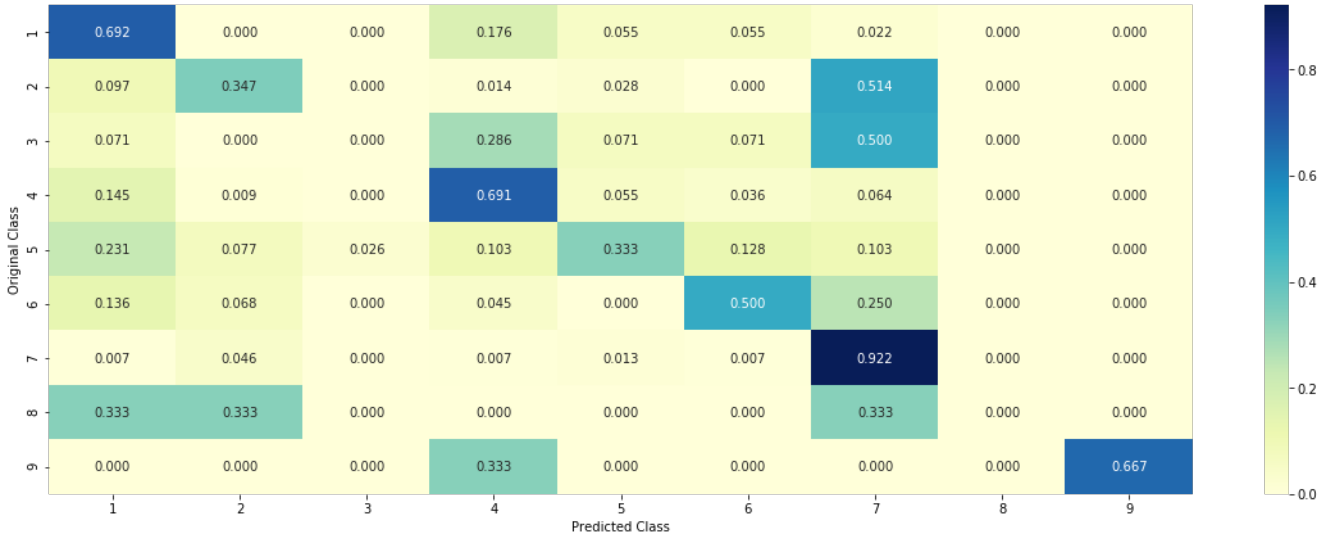
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
predict_and_plot_confusion_matrix(train_x_TFIDF, train_y, cv_x_TFIDF, cv_y, clf)
```

```
Log loss : 1.0982890881643197
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



---------- Precision matrix (Columm Sum=1) ----------

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



2.5 Random Forest Model

In [46]:

```
alpha = [100,200,500,1000,2000]
```

```
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_TFIDF, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_TFIDF, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_TFIDF)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[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 = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_TFIDF, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TFIDF, train_y)

predict_y = sig_clf.predict_proba(train_x_TFIDF)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss
is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_TFIDF)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss
is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_TFIDF)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

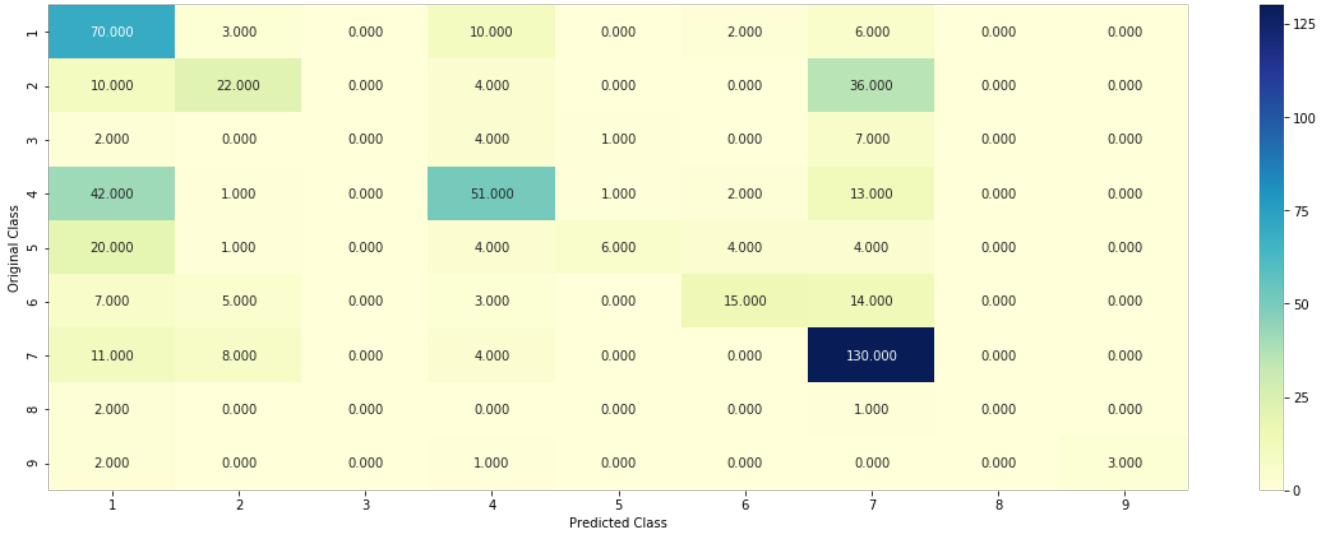
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2662209743857156
for n_estimators = 100 and max depth =  10
Log Loss : 1.2705696846379186
for n_estimators = 200 and max depth =  5
Log Loss : 1.2504264236753628
for n_estimators = 200 and max depth =  10
Log Loss : 1.2562393643321643
for n_estimators = 500 and max depth =  5
Log Loss : 1.2375998290634265
for n_estimators = 500 and max depth =  10
Log Loss : 1.2513113654902728
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2354336857278643
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2495736864822253
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2373493678664786
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2507570889611666
For values of best estimator =  1000 The train log loss is: 0.8400637782208904
For values of best estimator =  1000 The cross validation log loss is: 1.2354336857278643
For values of best estimator =  1000 The test log loss is: 1.1637274727011389
```

In [48]:

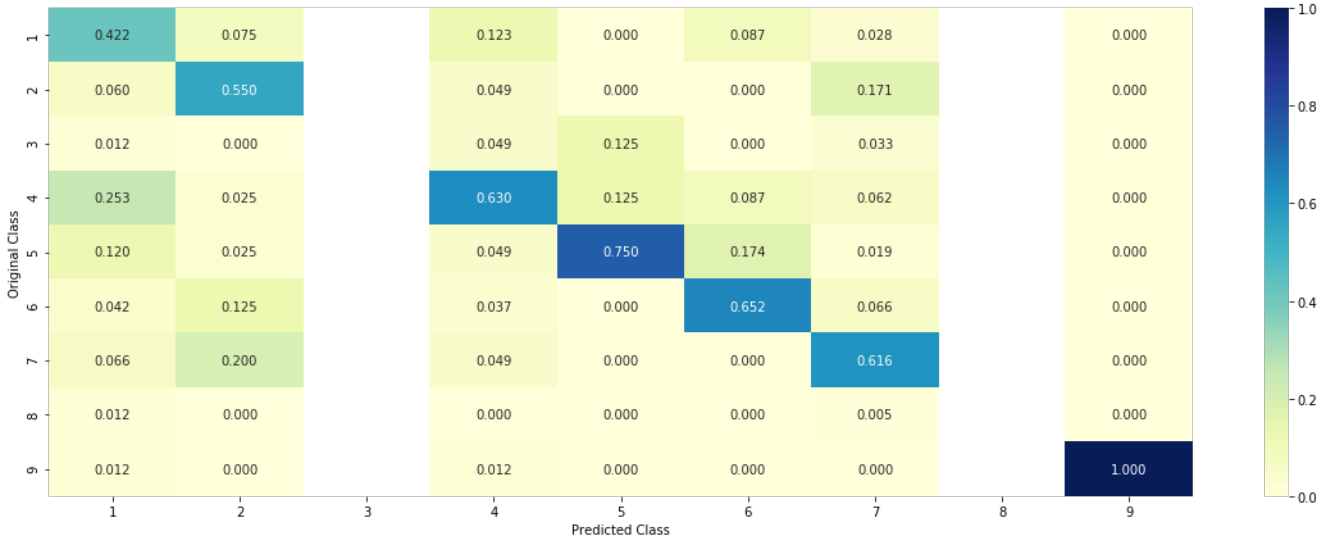
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_TFIDF, train_y,cv_x_TFIDF,cv_y, clf)
```

```
Log loss : 1.2354336857278643
```

```
Number of mis-classified points : 0.4417293233082707
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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

Apply Logistic regression with CountVectorizer Features, including both

unigrams and bigrams

In [41]:

```
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_ngram = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_ngram = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_ngram = variation_vectorizer.transform(cv_df['Variation'])
```

In [42]:

```
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_ngram = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_ngram = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_ngram = gene_vectorizer.transform(cv_df['Gene'])
```

In [48]:

```
print(train_gene_feature_ngram.shape)
print(train_variation_feature_ngram.shape)
```

```
(2124, 229)
(2124, 2061)
```

In [46]:

```
train_x_ngrams = hstack((train_gene_feature_ngram, train_variation_feature_ngram,
train_text_feature_TFIDF)).tocsr()
test_x_ngrams = hstack((test_gene_feature_ngram, test_variation_feature_ngram,
test_text_feature_TFIDF)).tocsr()
cv_x_ngrams = hstack((cv_gene_feature_ngram, cv_variation_feature_ngram,
cv_text_feature_TFIDF)).tocsr()

print("After stacking all 3 features using ngrams: ")
print(train_x_ngrams.shape,train_y.shape)
print(test_x_ngrams.shape,test_y.shape)
print(cv_x_ngrams.shape,cv_y.shape)
```

```
After stacking all 3 features using ngrams:
(2124, 3290) (2124,)
(665, 3290) (665,)
(532, 3290) (532,)
```

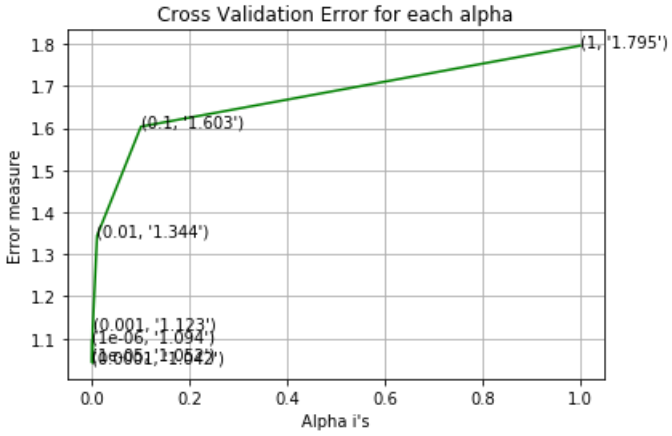
Logistic regression without class balancing

In [47]:

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_ngrams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngrams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngrams)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_ngrams, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_ngrams, train_y)

predict_y = sig_clf.predict_proba(train_x_ngrams)
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_x_ngrams)
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_x_ngrams)
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 = 1e-06
Log Loss : 1.0943408721223788
for alpha = 1e-05
Log Loss : 1.0517705885361721
for alpha = 0.0001
Log Loss : 1.041909431261045
for alpha = 0.001
Log Loss : 1.1234088972802792
for alpha = 0.01
Log Loss : 1.3441332105615478
for alpha = 0.1
Log Loss : 1.6034347141164957
for alpha = 1
Log Loss : 1.79548016764352
```

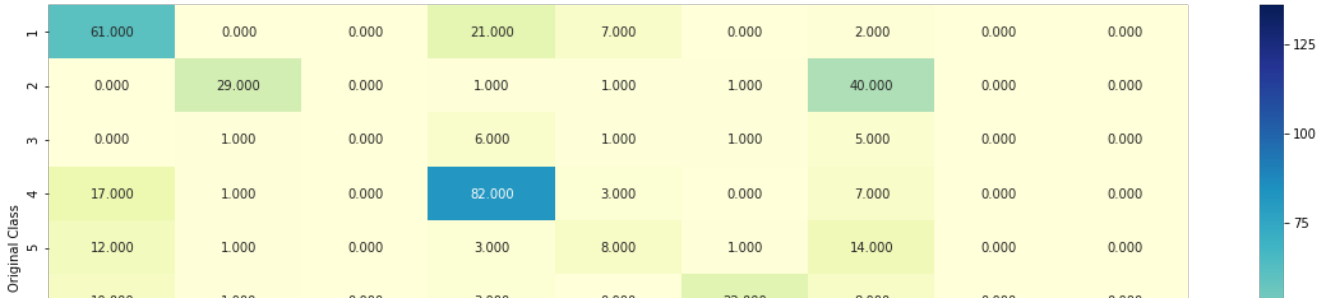


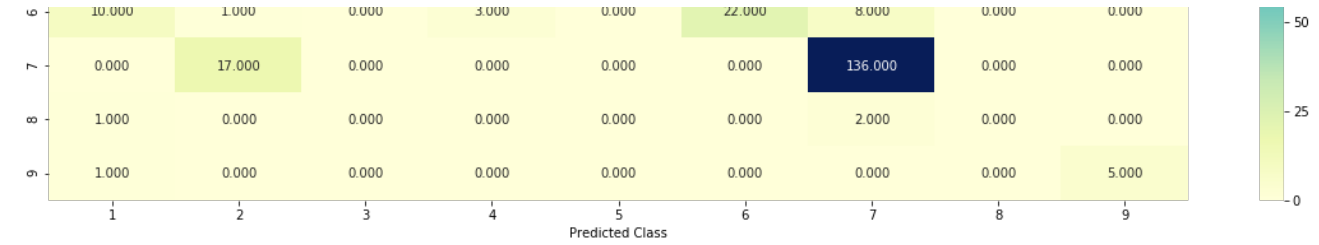

```
For values of best alpha =  0.0001 The train log loss is: 0.433962775138537
For values of best alpha =  0.0001 The cross validation log loss is: 1.041909431261045
For values of best alpha =  0.0001 The test log loss is: 0.953140830564431
```

In [49]:

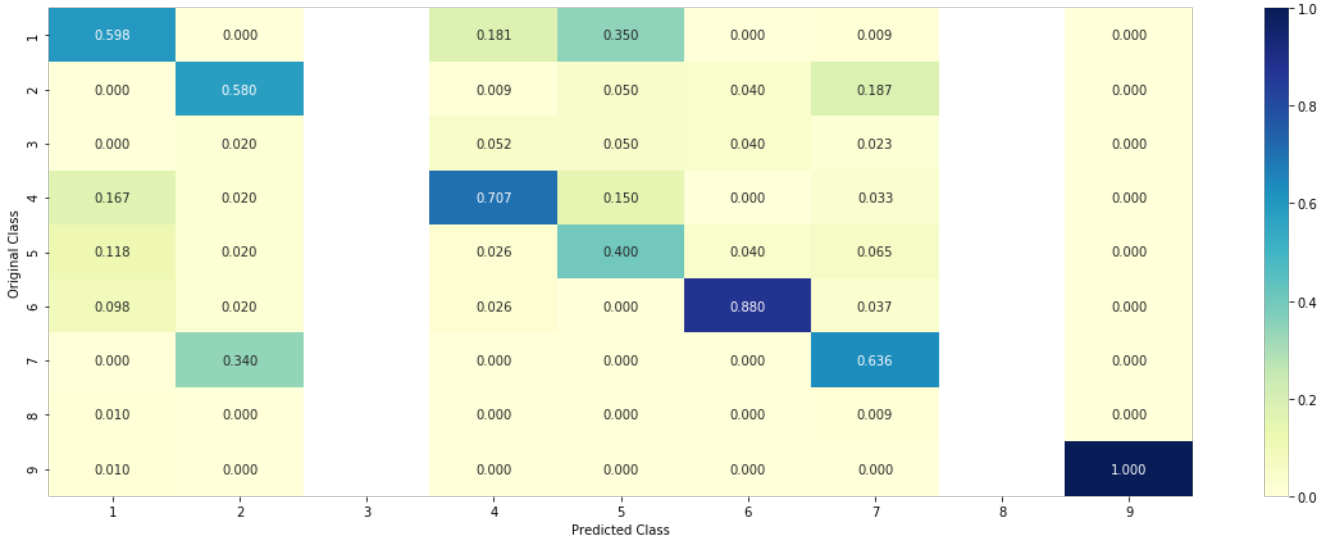
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_TFIDF, train_y, cv_x_TFIDF, cv_y, clf)
```

```
Log loss : 1.0409073096980948
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```

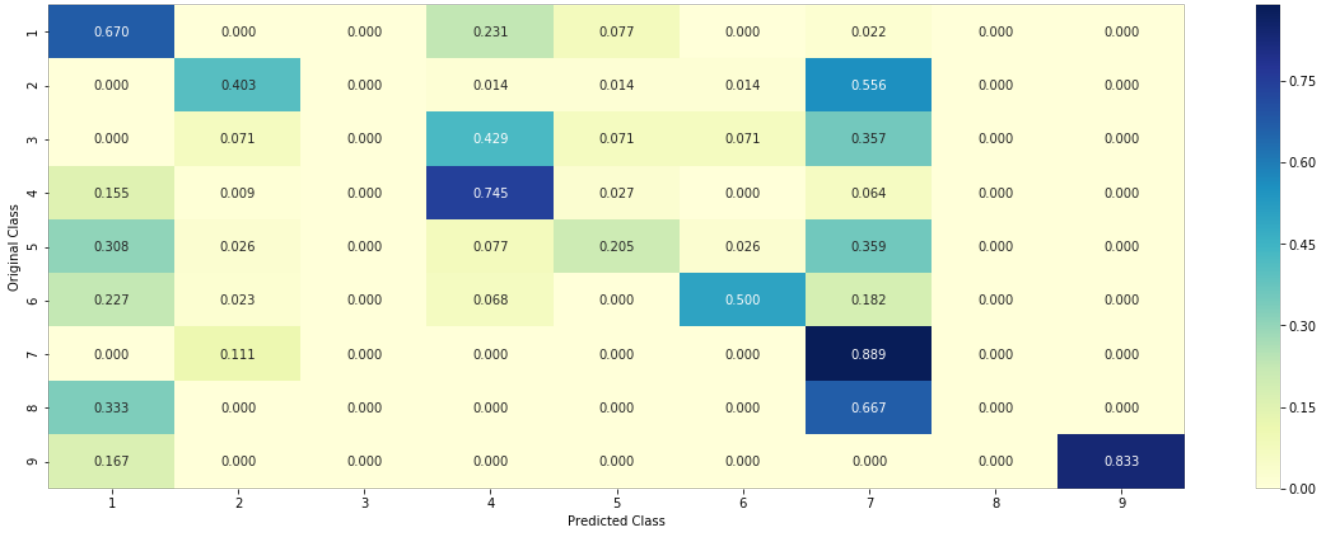





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



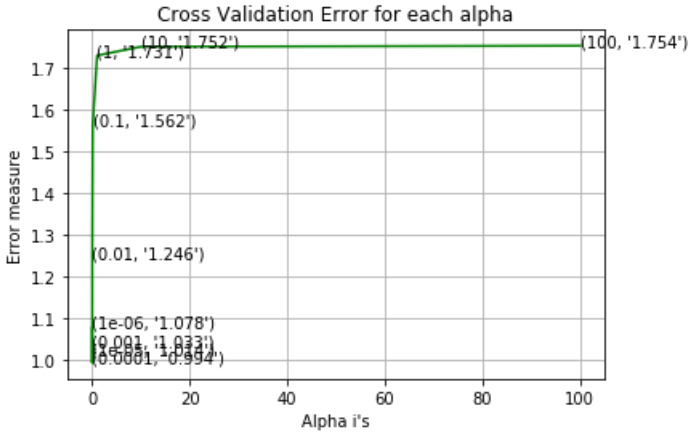
Logistic regression with class balancing

In [50]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_ngrams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngrams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngrams)
    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 = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_ngrams, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_ngrams, train_y)

predict_y = sig_clf.predict_proba(train_x_ngrams)
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_x_ngrams)
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_x_ngrams)
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 = 1e-06
Log Loss : 1.078244466375401
for alpha = 1e-05
Log Loss : 1.0140361595676115
for alpha = 0.0001
Log Loss : 0.9942847105984886
for alpha = 0.001
Log Loss : 1.0327489302279294
for alpha = 0.01
Log Loss : 1.2456843546285659
for alpha = 0.1
Log Loss : 1.5620699345595024
for alpha = 1
Log Loss : 1.730540987216278
for alpha = 10
Log Loss : 1.751503557403026
for alpha = 100
Log Loss : 1.75388126085341
```



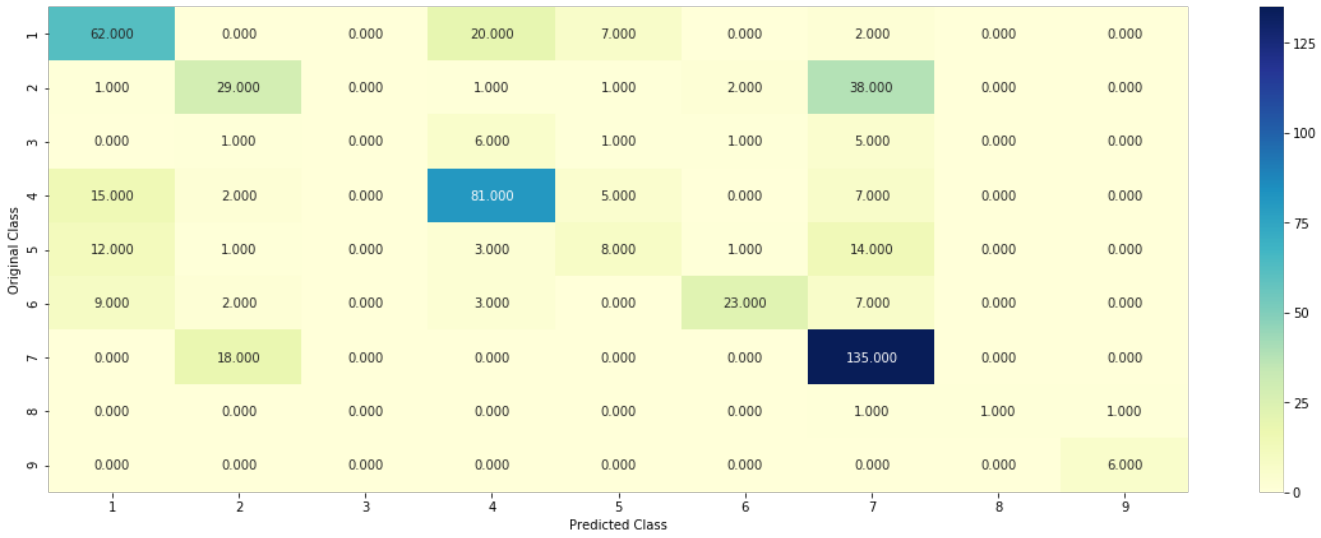

```
For values of best alpha =  0.0001 The train log loss is: 0.4438186274432016
For values of best alpha =  0.0001 The cross validation log loss is: 0.9942847105984886
For values of best alpha =  0.0001 The test log loss is: 0.918936916121376
```

In [51]:

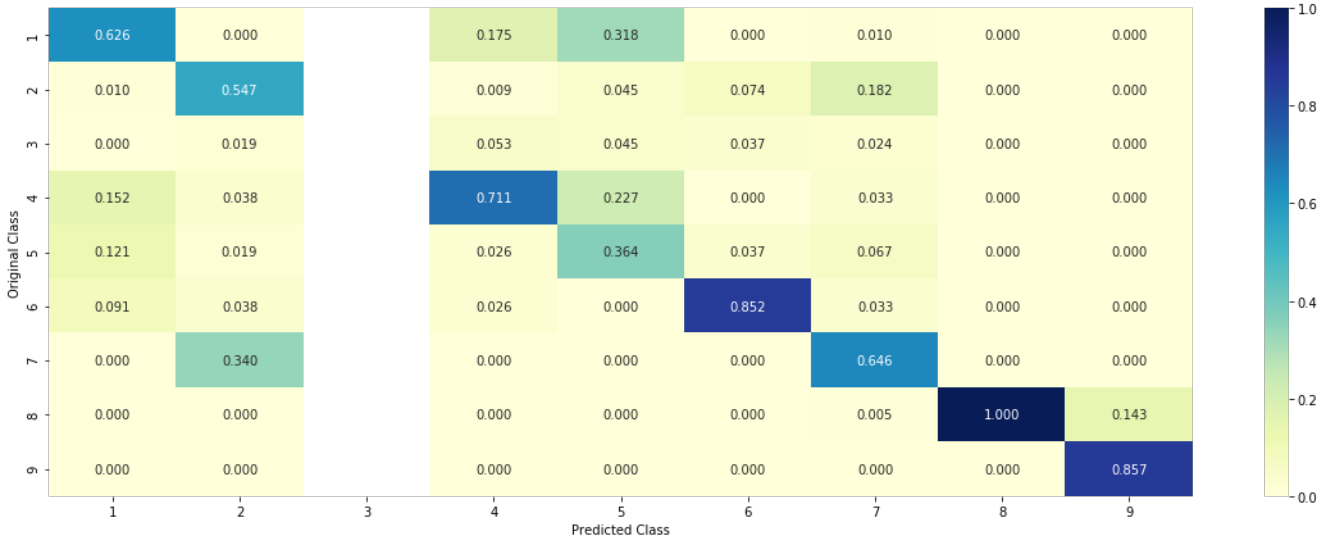
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
```

```
predict_and_plot_confusion_matrix(train_x_TFIDF, train_y, cv_x_TFIDF, cv_y, clf)
```

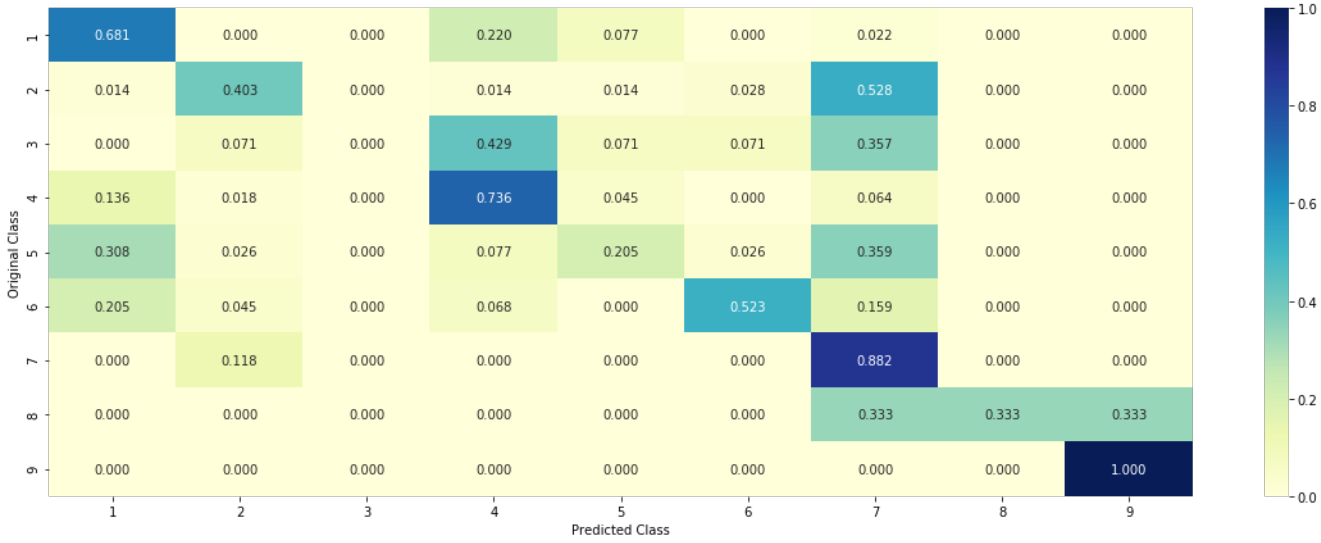
```
Log loss : 0.9929848697839546
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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

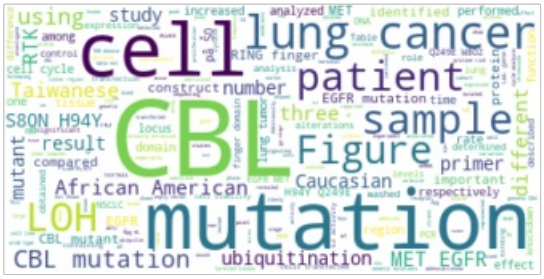


Advanced Feature Extraction of Text Feature

In [61]:

```
from wordcloud import WordCloud, STOPWORDS

text = result['TEXT'].values

stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")
stopwords.remove("no")
stopwords.remove("like")

wc = WordCloud(background_color="white", max_words=len(text[1]), stopwords=stopwords)
wc.generate(text[1])
print ("Word Cloud for Text data")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
Word Cloud for Text data
```



Adding Label Encoder, Binning Proba with Decision Tree

In [81]:

```
from sklearn.preprocessing import LabelEncoder

gle = LabelEncoder()
gene_labels = gle.fit_transform(result['Gene'])
variation_labels = gle.fit_transform(result['Variation'])
```

In [82]:

```
result['Gene_Label'] = gene_labels
result['Variation_Label'] = variation_labels
result.head()
```

Out[82]:

	ID	Gene	Variation	Class	TEXT	Gene_Label	Gene_tree	Variation_Label
0	0	FAM58A	Truncating_Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	85	0.200825	2629
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	39	0.096916	2856
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	39	0.096916	1897
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	39	0.096916	1667
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	39	0.096916	1447

In [83]:

```
tree_model = DecisionTreeClassifier(max_depth=2)
tree_model.fit(result.Gene_Label.to_frame(), y_true)
result['Gene_tree'] = tree_model.predict_proba(result.Gene_Label.to_frame())[:,1]

tree_model.fit(result.Variation_Label.to_frame(), y_true)
result['Variation_tree'] = tree_model.predict_proba(result.Variation_Label.to_frame())[:,1]

result.head()
```

Out[83]:

	ID	Gene	Variation	Class	TEXT	Gene_Label	Gene_tree	Variation_Label	Variation_tree
0	0	FAM58A	Truncating_Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	85	0.200825	2629	0.119423
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	39	0.096916	2856	0.119423
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	39	0.096916	1897	0.160169
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	39	0.096916	1667	0.160169
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	39	0.096916	1447	0.160169

In [84]:

```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# 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(result, 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 [87]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(train_df['Gene_tree'].values.reshape(-1,1))
train_df_gt = normalizer.transform(train_df['Gene_tree'].values.reshape(-1,1))
cv_df_gt = normalizer.transform(cv_df['Gene_tree'].values.reshape(-1,1))
test_df_gt = normalizer.transform(test_df['Gene_tree'].values.reshape(-1,1))
```

In [88]:

```
normalizer.fit(train_df['Gene_Label'].values.reshape(-1,1))
train_df_gl = normalizer.transform(train_df['Gene_Label'].values.reshape(-1,1))
cv_df_gl = normalizer.transform(cv_df['Gene_Label'].values.reshape(-1,1))
test_df_gl = normalizer.transform(test_df['Gene_Label'].values.reshape(-1,1))
```

In [89]:

```
normalizer.fit(train_df['Variation_Label'].values.reshape(-1,1))
train_df_vl = normalizer.transform(train_df['Variation_Label'].values.reshape(-1,1))
cv_df_vl = normalizer.transform(cv_df['Variation_Label'].values.reshape(-1,1))
test_df_vl = normalizer.transform(test_df['Variation_Label'].values.reshape(-1,1))
```

In [90]:

```
normalizer.fit(train_df['Variation_tree'].values.reshape(-1,1))
train_df_vt = normalizer.transform(train_df['Variation_tree'].values.reshape(-1,1))
cv_df_vt = normalizer.transform(cv_df['Variation_tree'].values.reshape(-1,1))
```

```
test_df_vt = normalizer.transform(test_df['Variation_tree'].values.reshape(-1,1))
```

In [92]:

```
train_x_adv = hstack((train_x_ngrams,train_df_gt,train_df_gl,train_df_vt,train_df_vl)).tocsr()
test_x_adv = hstack((test_x_ngrams,test_df_gt,test_df_gl,test_df_vt,test_df_vl)).tocsr()
cv_x_adv = hstack((cv_x_ngrams,cv_df_gt,cv_df_gl,cv_df_vt,cv_df_vl)).tocsr()

print("After stacking advance features : ")
print(train_x_adv.shape,train_y.shape)
print(test_x_adv.shape,test_y.shape)
print(cv_x_adv.shape,cv_y.shape)
```

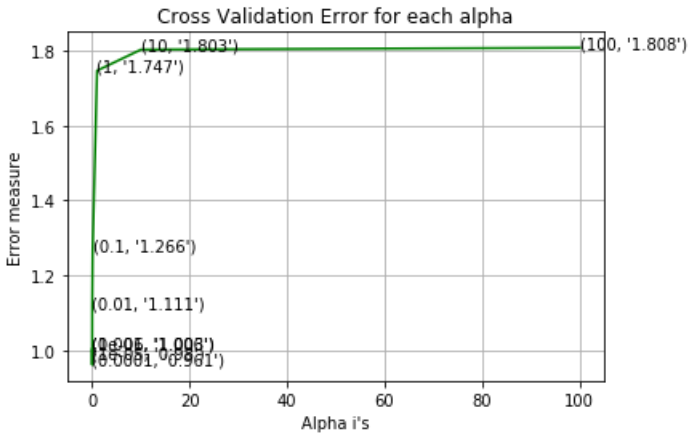
```
After stacking advance features :
(2124, 3294) (2124,)
(665, 3294) (665,)
(532, 3294) (532,)
```

In [97]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_adv, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_adv, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_adv)
    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_adv = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log'
, random_state=42)
clf_adv.fit(train_x_adv, train_y)
sig_clf_adv = CalibratedClassifierCV(clf_adv, method="sigmoid")
sig_clf_adv.fit(train_x_adv, train_y)

predict_y = sig_clf_adv.predict_proba(train_x_adv)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,
predict_y, labels=clf_adv.classes_, eps=1e-15))
predict_y = sig_clf_adv.predict_proba(cv_x_adv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(cv_y, predict_y, labels=clf_adv.classes_, eps=1e-15))
predict_y = sig_clf_adv.predict_proba(test_x_adv)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, p
redict_y, labels=clf_adv.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.0028680603900182
for alpha = 1e-05
Log Loss : 0.9800686292834285
for alpha = 0.0001
Log Loss : 0.9610753376010008
for alpha = 0.001
Log Loss : 1.005706119748458
for alpha = 0.01
Log Loss : 1.1111274422056476
for alpha = 0.1
Log Loss : 1.2663909977647192
```

```
Log Loss : 1.2665909977647192
for alpha = 1
Log Loss : 1.7466850825349969
for alpha = 10
Log Loss : 1.8027253685999163
for alpha = 100
Log Loss : 1.8078292264759115
```




```
For values of best alpha =  0.0001 The train log loss is: 0.45849341099848606
For values of best alpha =  0.0001 The cross validation log loss is: 0.9610753376010008
For values of best alpha =  0.0001 The test log loss is: 0.9177706464701678
```

In [105]:

```
pred_y = sig_clf_adv.predict(test_x_adv)
print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
```

```
Number of mis-classified points : 0.3263157894736842
```

In [107]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model","Vectorizer","Train log loss","Test log loss","Misclassified"]

x.add_row(["Naive Bayes","One Hot coding + TFIDF",0.527,1.2,36.09])
x.add_row(["LR with class balancing","One Hot coding + TFIDF",0.451,1.02,34.02])
x.add_row(["LR without class balancing","One Hot coding + TFIDF",0.447,1.047,34.96])
x.add_row(["Linear SVM","One Hot coding + TFIDF",0.405,1.052,35.33])
x.add_row(["Random Forest","One Hot coding + TFIDF",0.84,1.163,44.17])
x.add_row(["LR without class balancing","OHC+ Bigram + TFIDF",0.433,0.953,35.55])
x.add_row(["LR with class balancing","OHC+ Bigram +TFIDF",0.443,0.918,35.15])
x.add_row(["LR class balancing","Binning+Bigram+TFIDF",0.458,0.917,32.63])

print(x)
```

```
+---------------------------+------------------------+----------------+---------------+-----------
-+
|           Model           |       Vectorizer       | Train log loss | Test log loss | Misclassif
ed |
+---------------------------+------------------------+----------------+---------------+-----------
-+
|        Naive Bayes        | One Hot coding + TFIDF |     0.527      |      1.2      |     36.09
|
|  LR with class balancing  | One Hot coding + TFIDF |     0.451      |      1.02     |     34.02
|
| LR without class balancing | One Hot coding + TFIDF |     0.447      |     1.047     |     34.96
|
|         Linear SVM        | One Hot coding + TFIDF |     0.405      |     1.052     |     35.33
|
|       Random Forest       | One Hot coding + TFIDF |      0.84      |     1.163     |     44.17
|
| LR without class balancing |  OHC+ Bigram + TFIDF   |     0.433      |     0.953     |     35.55
|
|  LR with class balancing  |  OHC+ Bigram +TFIDF    |     0.443      |     0.918     |     35.15
```

| LR with class balancing | Chi2 Bigram TFIDF | 0.445 | 0.918 | 38.13 |
| LR class balancing | Binning+Bigram+TFIDF | 0.458 | 0.917 | 32.63 |