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.cross_validation 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
```

```
/home/chitra/.local/lib/python3.6/site-packages/matplotlib/__init__.py:886:
MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative to the 'datapath'
directory.
  "found relative to the 'datapath' directory.".format(key))
```

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
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...

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...

	ID	Gene	Variation	Class	TEXT
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.shape
```

Out[5]:

```
(3321, 5)
```

In [6]:

```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3321 entries, 0 to 3320
Data columns (total 5 columns):
ID            3321 non-null int64
Gene          3321 non-null object
Variation     3321 non-null object
Class         3321 non-null int64
TEXT          3316 non-null object
dtypes: int64(2), object(3)
memory usage: 155.7+ KB
```

In [7]:

```
result["Variation"].describe()
```

Out[7]:

```
count                     3321
unique                    2996
top       Truncating Mutations
freq                        93
Name: Variation, dtype: object
```

In [8]:

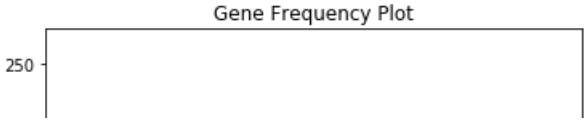
```
result["Gene"].describe()
```

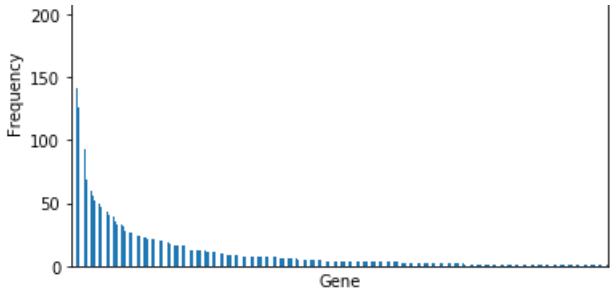
Out[8]:

```
count      3321
unique      264
top       BRCA1
freq        264
Name: Gene, dtype: object
```

In [9]:

```
## gene frequency plot
plt.figure()
ax = result['Gene'].value_counts().plot(kind='bar')

ax.get_xaxis().set_ticks([])
ax.set_title('Gene Frequency Plot')
ax.set_xlabel('Gene')
ax.set_ylabel('Frequency')

plt.show()
```

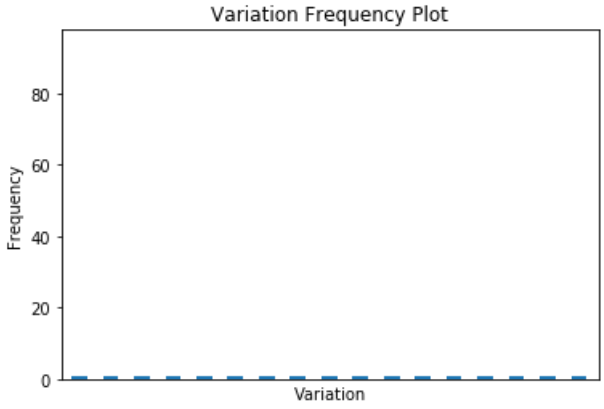




Observation: Gene contains minimum independent unique values.

In [10]:

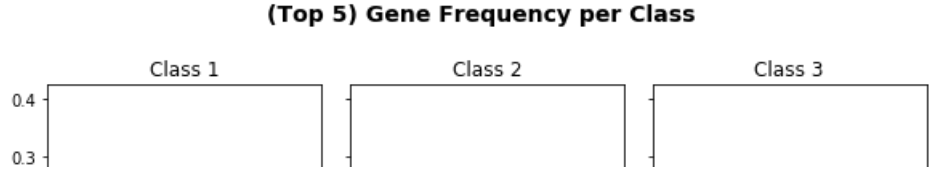
```
## gene frequency plot
plt.figure()
ax = result['Variation'].value_counts().plot(kind='bar')

ax.get_xaxis().set_ticks([])
ax.set_title('Variation Frequency Plot')
ax.set_xlabel('Variation')
ax.set_ylabel('Frequency')

plt.show()
```

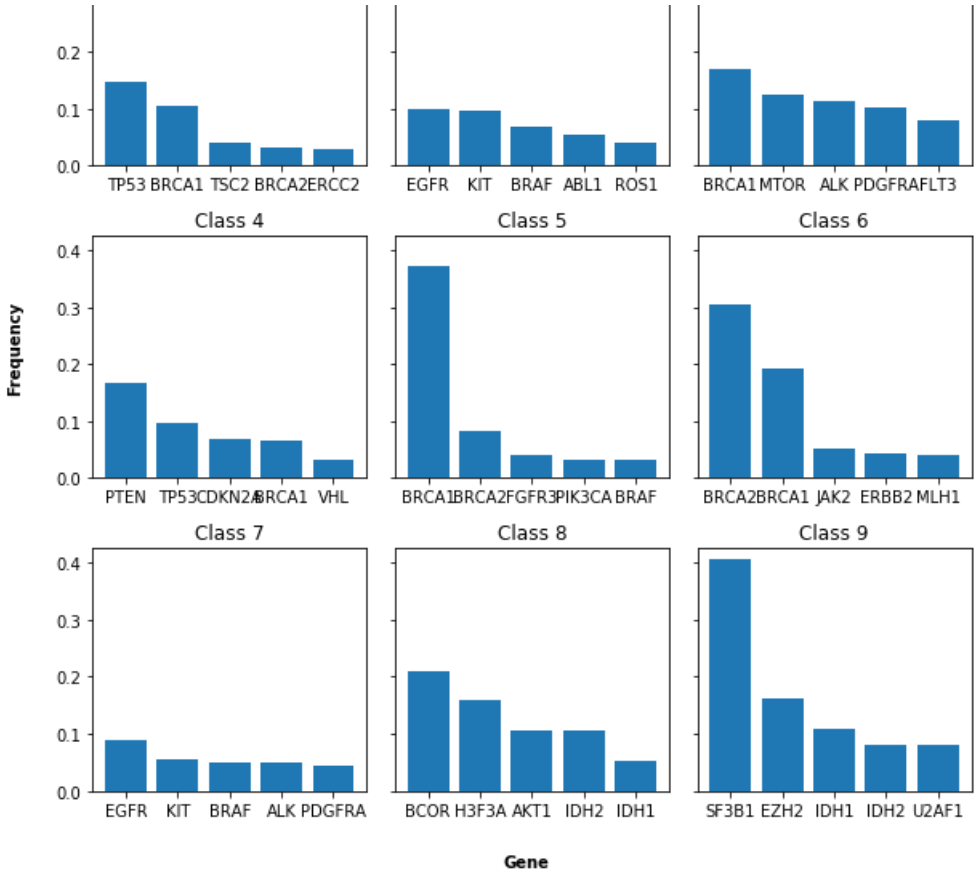


Observation: Variation contains maximum independent unique values.

In [11]:

```
fig, axes = plt.subplots(nrows=3, ncols=3, sharey=True, figsize=(9,9))

# Normalize value counts for better comparison
def normalize_group(x):
    label, repetition = x.index, x
    t = sum(repetition)
    r = [n/t for n in repetition]
    return label, r

for idx, g in enumerate(result.groupby('Class')):
    label, val = normalize_group(g[1]["Gene"].value_counts())
    ax = axes.flat[idx]
    ax.bar(np.arange(5), val[:5],
           tick_label=label[:5])
    ax.set_title("Class {}".format(g[0]))

fig.text(0.5, 0.97, '(Top 5) Gene Frequency per Class', ha='center', fontsize=14, fontweight='bold'
)
fig.text(0.5, 0, 'Gene', ha='center', fontweight='bold')
fig.text(0, 0.5, 'Frequency', va='center', rotation='vertical', fontweight='bold')
fig.tight_layout(rect=[0.03, 0.03, 0.95, 0.95])
```





In [12]:

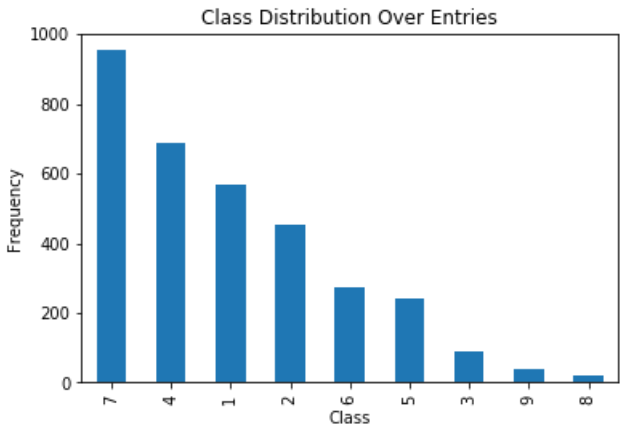
```
result.Class.unique()
```

Out[12]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [13]:

```
#### class distribution
plt.figure()
ax = result['Class'].value_counts().plot(kind='bar')

ax.set_title('Class Distribution Over Entries')
ax.set_xlabel('Class')
ax.set_ylabel('Frequency')

plt.show()
```



Observation: Class 8 and class 9 have less examples.

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'] = str(result['Gene'] +' '+result['Variation'])
```

In [7]:

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

Out[7]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	0 FAM58A Truncating Mutations\n1 ...

Feature Engineering

In [ ]:

```
#### https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/
#### https://www.kaggle.com/shivamb/extensive-text-data-feature-engineering
```

In [9]:

```
### Gene_Variation feature
result['Gene_Variation'] = result['Gene'] + " " + result["Variation"]
result.head()
```

Out[9]:

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

In [10]:

```
result['Gene_Share'] = result.apply(lambda r: sum([1 for w in r['Gene'].split() if w in r['TEXT'].split()]), axis=1)
result.head()
```

Out[10]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc	CBL W802*	1
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1

In [11]:

```
result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['Variation'].split(' ') if w i
n r['TEXT'].split(' ')]), axis=1)
result["Variation_Share"].value_counts()
```

Out[11]:

```
1    1673
0    1576
2      58
3      10
5       2
4       2
Name: Variation_Share, dtype: int64
```

In [12]:

```
### Number of words
result["Word_Count"] = result["TEXT"].apply(lambda x: len(x.split()))
result.head()
```

Out[12]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5756
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5756
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202

In [13]:

```
result["Word_Count_5000"] = result["Word_Count"].apply(lambda x: 1 if x > 5000 else 0)
result.head()
```

Out[13]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5756	1
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5756	1
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1
4	4	CBL	L399V	4	Oncogenic					

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	
4	4	CBL	L399V	4	mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1	

In [14]:

```
### Number of characters
result['Character_Count'] = result['TEXT'].apply(lambda x: len(str(x)))
result.head()
```

Out[14]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Charac
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1	
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5756	1	
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5756	1	
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1	
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1	

In [15]:

```
### Average word length
result['Avg_length'] = result['Character_Count'] / result['Word_Count']
result.head()
```

Out[15]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Charac
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1	
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5756	1	
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5756	1	
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1	
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1	

3.1.3. Preprocessing of text

In [16]:

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


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

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

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

In [17]:

```
#text processing stage.
start_time = time.clock()
for index, row in result.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")
```

```
Time took for preprocessing the text : 181.121241 seconds
```

In [18]:

```
result.head()
```

Out[18]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Charac
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1	
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5756	1	
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5756	1	
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1	
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1	

ID	Gene	Variation	Class	TEXT	Gene Variation	Gene Share	Variation Share	Word Count	Word Count 5000	Charac

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 [19]:

```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
result.Gene_Variation = result.Gene_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 [20]:

```
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])
```

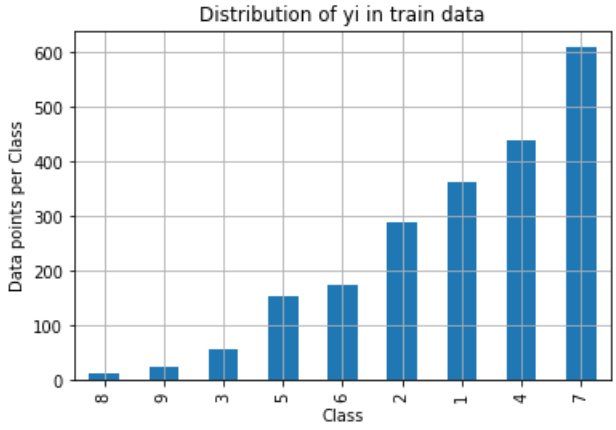
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

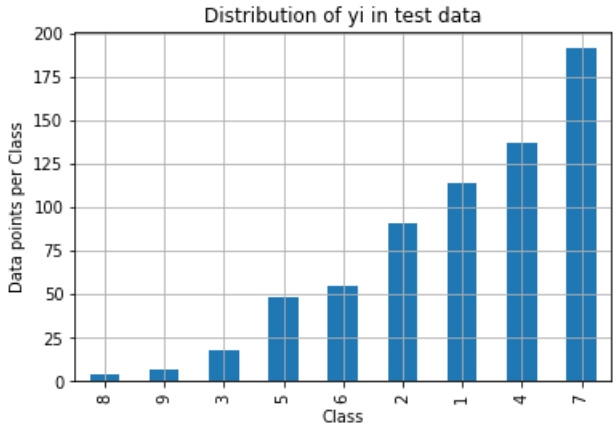
In [21]:

```
# 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().sort_values()
test_class_distribution = test_df['Class'].value_counts().sort_values()
cv_class_distribution = cv_df['Class'].value_counts().sort_values()

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), '%)')
```

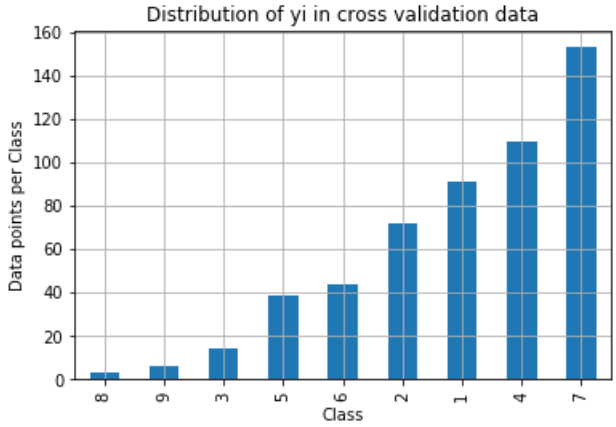



```
Number of data points in class 9 : 609 ( 28.672 %)
Number of data points in class 8 : 439 ( 20.669 %)
Number of data points in class 7 : 363 ( 17.09 %)
Number of data points in class 6 : 289 ( 13.606 %)
Number of data points in class 5 : 176 ( 8.286 %)
Number of data points in class 4 : 155 ( 7.298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




```
Number of data points in class 9 : 191 ( 28.722 %)
Number of data points in class 8 : 137 ( 20.602 %)
Number of data points in class 7 : 114 ( 17.143 %)
Number of data points in class 6 : 91 ( 13.684 %)
Number of data points in class 5 : 55 ( 8.271 %)
Number of data points in class 4 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 2 : 7 ( 1.053 %)
```

```
Number of data points in class 2 : 7 ( 1.053 %)
Number of data points in class 1 : 4 ( 0.602 %)
--------------------------------------------------------------------------------
```




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

3.2 Prediction using a 'Random' Model

In [22]:

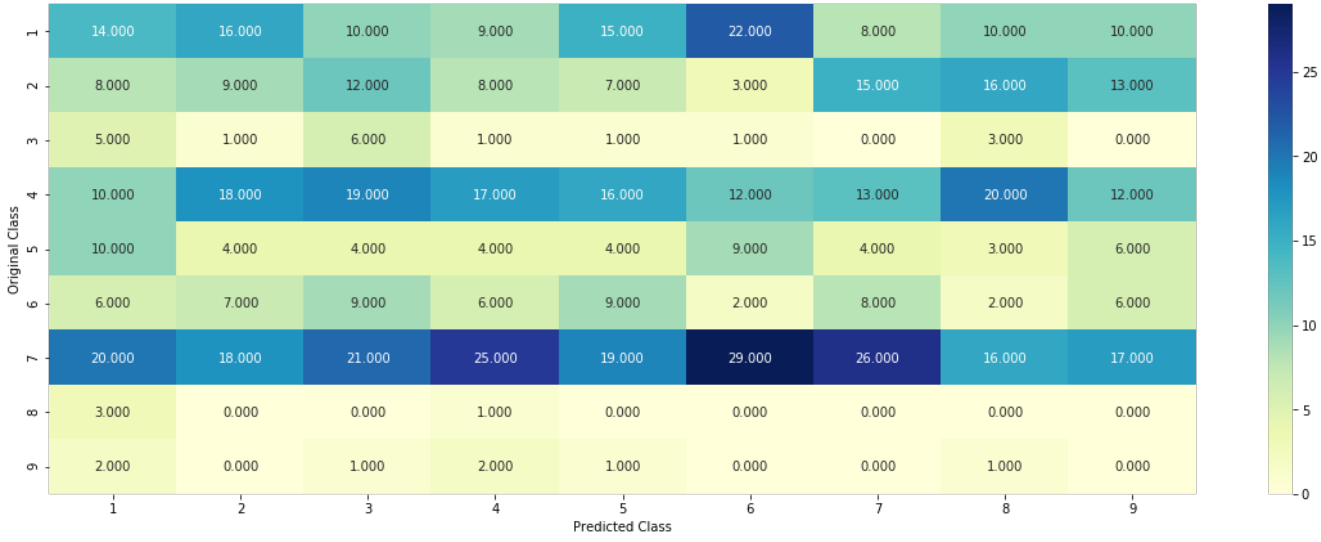
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Column 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()
```

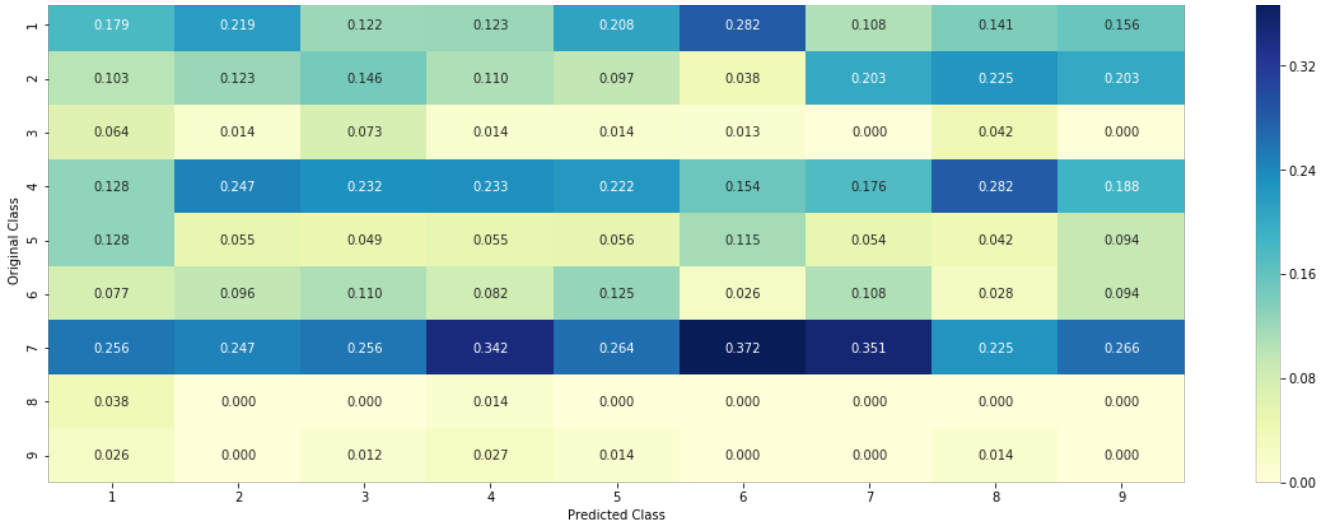
In [23]:

```
# 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)
```

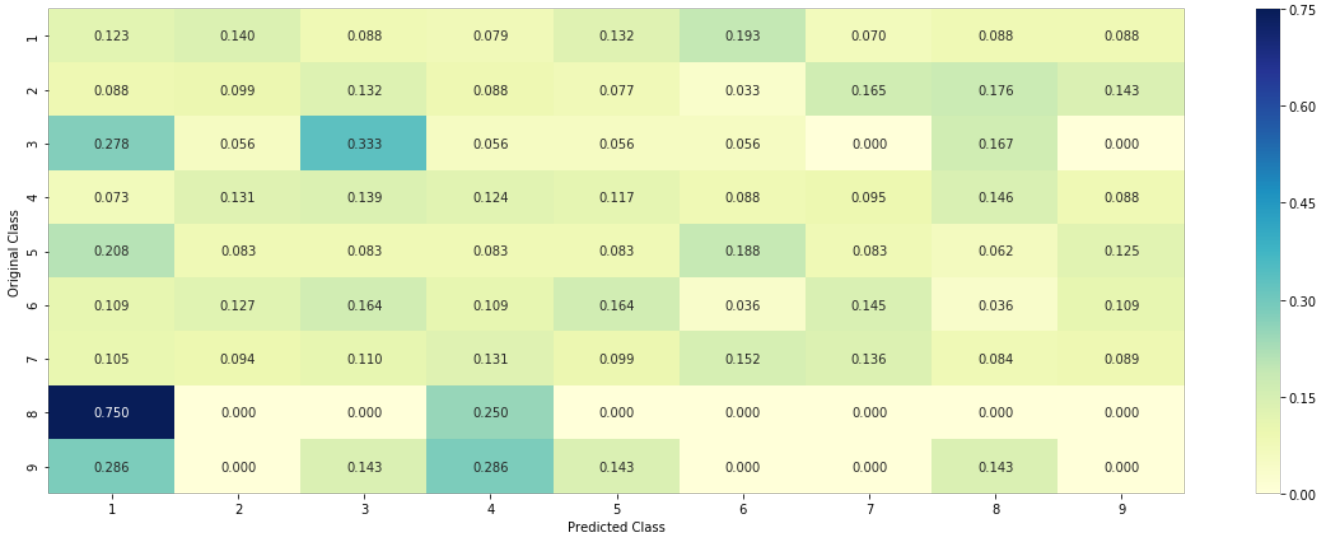
```
Log loss on Cross Validation Data using Random Model 2.480851023499652
Log loss on Test Data using Random Model 2.45595752446028
-------------------- Confusion matrix --------------------
```




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




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



3.3 Univariate Analysis

In [24]:

```
# 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': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
```

```
    #     }
    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 [25]:

```
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 : 238
BRCA1     176
TP53      110
EGFR      100
PTEN       80
BRCA2      80
BRAF       58
KIT        58
ERBB2      48
ALK        44
PDGFRA     39
Name: Gene, dtype: int64
```

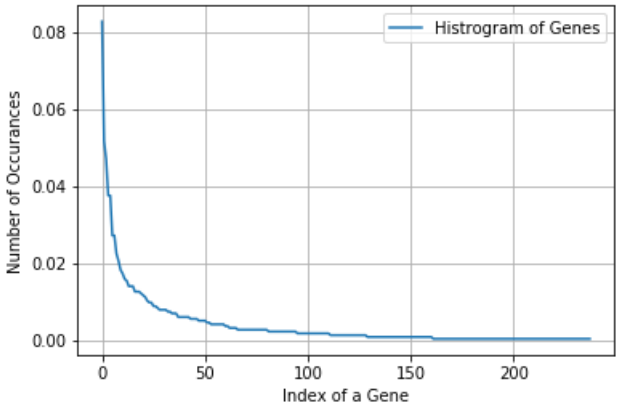
In [26]:

```
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 238 different categories of genes in the train data, and they are distibuted as fol
lows
```

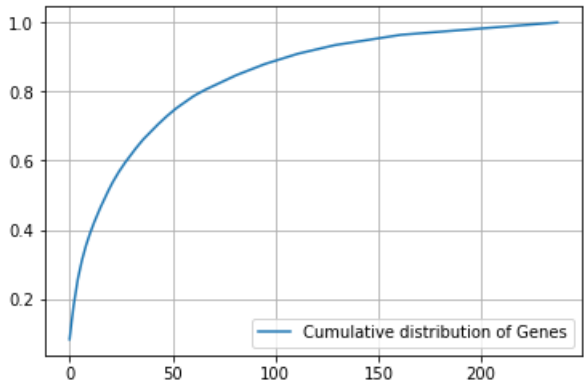
In [27]:

```
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 [28]:

```
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 [29]:

```
#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 [30]:

```
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)
```

one feature: (2124, 9)

In [31]:

```
# 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 [32]:

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

Out[32]:

```
1196    PIK3CA
2816     BRCA2
2592     BRCA1
2794     BRCA2
1642      FLT3
Name: Gene, dtype: object
```

In [33]:

```
gene_vectorizer.get_feature_names()
```

Out[33]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk8',
```

```
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf3',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
```

```
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'rras2',
```

```
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [34]:

```
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, 237)
```

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

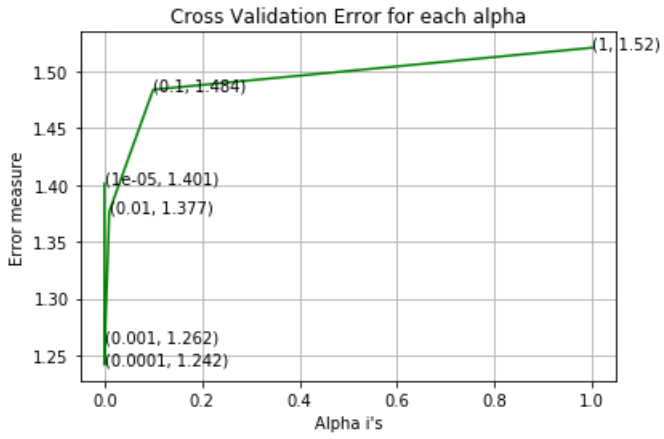
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [35]:

```
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: 1.4009901785923133
For values of alpha =  0.0001 The log loss is: 1.241739648284663
For values of alpha =  0.001 The log loss is: 1.262011689140139
For values of alpha =  0.01 The log loss is: 1.376634449399032
For values of alpha =  0.1 The log loss is: 1.4836179266325569
For values of alpha =  1 The log loss is: 1.5204330300921665
```




```
For values of best alpha =  0.0001 The train log loss is: 1.031174571325565
For values of best alpha =  0.0001 The cross validation log loss is: 1.241739648284663
For values of best alpha =  0.0001 The test log loss is: 1.1908614819381689
```

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 [36]:

```
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  238  genes in train dataset?
Ans
1. In test data 649 out of 665 : 97.59398496240601
2. In cross validation data 517 out of  532 : 97.18045112781954
```

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 [37]:

```
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 : 1931
Truncating_Mutations    59
Amplification           45
Deletion                42
Fusions                 25
G12V                     4
Overexpression           4
Q61H                     3
Q61L                     3
Y42C                     2
I31M                     2
Name: Variation, dtype: int64
```

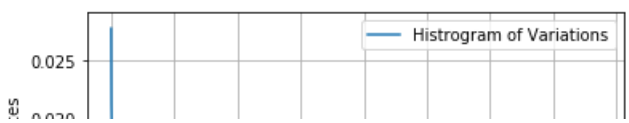
In [38]:

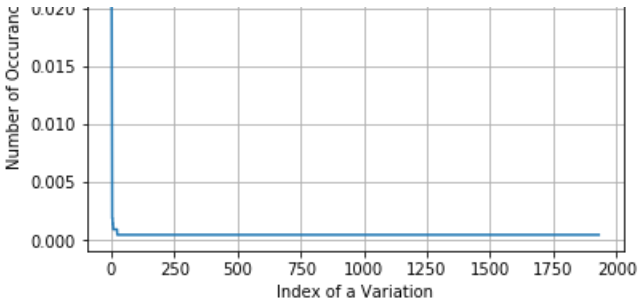
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1931 different categories of variations in the train data, and they are distibuted
as follows
```

In [39]:

```
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()
```

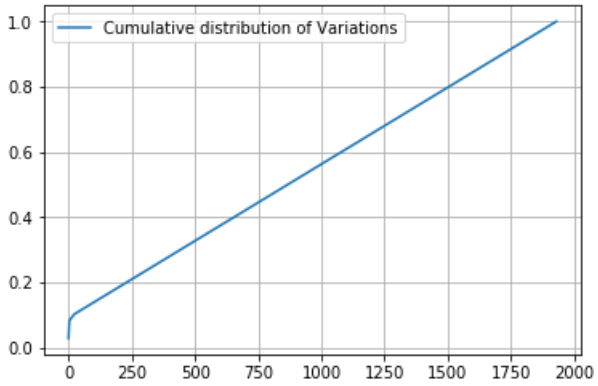




In [40]:

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

```
[0.02777778 0.04896422 0.06873823 ... 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 [41]:

```
# 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 [42]:

```
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 [43]:

In [43]:

```
# 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 [44]:

```
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, 1959)
```

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

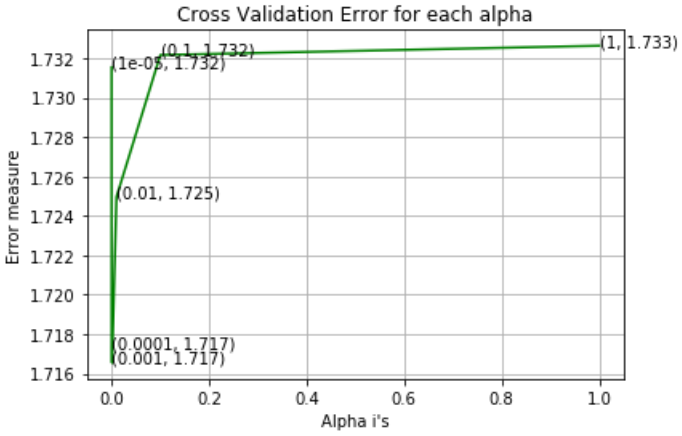
Let's build a model just like the earlier!

In [45]:

```
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: 1.7315379309586805
For values of alpha =  0.0001 The log loss is: 1.7173216960395492
For values of alpha =  0.001 The log loss is: 1.7165415283997856
For values of alpha =  0.01 The log loss is: 1.7249600194359764
For values of alpha =  0.1 The log loss is: 1.732156934783147
For values of alpha =  1 The log loss is: 1.7326278187463893
```




```
For values of best alpha =  0.001 The train log loss is: 1.0391602271787637
For values of best alpha =  0.001 The cross validation log loss is: 1.7165415283997856
For values of best alpha =  0.001 The test log loss is: 1.698076489503657
```

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 [46]:

```
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  1931  genes in test and cross validation data
sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 54 out of  532 : 10.150375939849624
```

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 [47]:

```
# 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 [48]:

```
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 [50]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
from sklearn.feature_extraction.text import TfidfVectorizer
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
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 : 1000
```

In [51]:

```
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 [52]:

```
#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 [53]:

```
# 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 [54]:

```
# 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 [55]:

```
#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 [56]:

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

```
Counter({902.5458184404599: 1, 741.5696828587056: 1, 561.5044988512566: 1, 549.25151913308: 1,
335.5554203263332: 1, 231.11184140225387: 1, 173.88625172282056: 1, 167.2496442560766: 1,
161.98543166333982: 1, 158.44181284063282: 1, 128.891156972169: 1, 125.30296753402489: 1,
124.73220247185635: 1, 118.58552771699092: 1, 113.28960803926495: 1, 100.70147769904173: 1,
90.53559595053224: 1, 90.0555443113816: 1, 88.39000840897897: 1, 80.81062477554326: 1,
80.0533056100129: 1, 79.7531141095276: 1, 78.93371576844366: 1, 72.63762902496197: 1,
70.91338032843682: 1, 70.7880556612402: 1, 70.58507861752315: 1, 70.18969214773344: 1,
67.85748729402812: 1, 67.60617317066804: 1, 67.50585582239239: 1, 66.31602908463708: 1,
62.978956699497516: 1, 59.11133151918798: 1, 57.979224575636955: 1, 57.26828078104574: 1,
56.731747564513505: 1, 55.00345078062726: 1, 54.62498422042046: 1, 53.426086767218706: 1,
51.069513531911696: 1, 50.19318919716382: 1, 49.50526836754021: 1, 48.53034157748338: 1, 48.3858816
6622585: 1, 47.65525653941947: 1, 46.56381479458002: 1, 46.453671916379115: 1, 43.512283374418324:
1, 43.26308509485172: 1, 41.840476751177626: 1, 41.13059936248549: 1, 40.383918596343555: 1,
39.321857642101875: 1, 39.189178071744294: 1, 39.10514878680281: 1, 39.06872039831969: 1,
38.824965302782104: 1, 38.63799603224563: 1, 38.35016750457289: 1, 37.92202761087895: 1, 37.0623157
1718099: 1, 36.830416663265: 1, 35.61010041702103: 1, 35.39729001234439: 1, 34.54585630191552: 1,
33.84647253748432: 1, 32.73507916556703: 1, 32.69778654484788: 1, 31.341990087808092: 1,
31.267352883603433: 1, 31.217083724876627: 1, 30.158426665364285: 1, 29.817181515780753: 1,
29.75588850770646: 1, 29.43417398191592: 1, 29.118901145096068: 1, 28.401210558734636: 1,
28.353367494974357: 1, 28.208532876008494: 1, 27.946237757145173: 1, 27.902410381551412: 1,
27.819411198835116: 1, 27.772213321816402: 1, 27.69909407967305: 1, 27.687212843079074: 1,
27.530866027526297: 1, 27.14327069291431: 1, 26.926053675975375: 1, 26.719923668170274: 1,
26.6908273041871: 1, 26.48359161716056: 1, 26.452945763979315: 1, 26.378493590578486: 1,
26.039546520613357: 1, 25.816787123045334: 1, 25.51489586250253: 1, 25.412417819528024: 1,
25.330987034794564: 1, 25.116115931690864: 1, 25.088592759460944: 1, 25.00065799720309: 1,
24.89652263806983: 1, 24.74360359156307: 1, 24.71456133580431: 1, 24.69772826768: 1,
24.523809426486075: 1, 24.398306654781955: 1, 24.13396574317239: 1, 24.082987197103925: 1,
```

23.70571539605116: 1, 23.628276270592167: 1, 23.54294338573282: 1, 23.198769417453494: 1,
23.110484975616938: 1, 23.00930104074548: 1, 22.86280609831157: 1, 22.659307689402368: 1,
22.64363866950496: 1, 22.624867892698195: 1, 22.205235643258458: 1, 22.08293846151838: 1,
22.065795503157762: 1, 22.048770824046642: 1, 21.933846550501702: 1, 21.882165983283045: 1,
21.677371562630846: 1, 21.653701045654465: 1, 21.281088725792234: 1, 21.075132192425706: 1,
21.056976165216255: 1, 20.90670236688227: 1, 20.8871131567222: 1, 20.753392893922886: 1,
20.599843340848803: 1, 20.441780587162476: 1, 20.412384608178726: 1, 20.389610974816488: 1,
20.13994129979737: 1, 20.125688742350533: 1, 19.80531840909242: 1, 19.755634828635483: 1,
19.743908760921315: 1, 19.536606439349942: 1, 19.47348707350421: 1, 19.39819210968346: 1,
19.253464552832334: 1, 19.21964240063173: 1, 19.20371178994851: 1, 19.12022228156036: 1, 19.0674716
14644273: 1, 19.031661638257653: 1, 19.010382493032417: 1, 18.940667463793254: 1,
18.905714384720227: 1, 18.800168447983392: 1, 18.769287752087898: 1, 18.767456369091292: 1,
18.750774308260713: 1, 18.746559078281084: 1, 18.70845521690482: 1, 18.64833902359023: 1,
18.630112730464997: 1, 18.618541341887575: 1, 18.53307462194202: 1, 18.529941865479778: 1,
18.505322076090664: 1, 18.492394328475527: 1, 18.434879529159783: 1, 18.259039785711085: 1,
18.114731030941392: 1, 18.053741958298673: 1, 18.003779920070027: 1, 17.94048999240867: 1,
17.846065898300726: 1, 17.828138726657464: 1, 17.7966997749118: 1, 17.685003584518494: 1,
17.64612077055493: 1, 17.50181044642015: 1, 17.466100269507265: 1, 17.458728064973485: 1,
17.441107922786063: 1, 17.38112453970289: 1, 17.30901536134392: 1, 17.275545616271941: 1, 17.0734862
57777226: 1, 17.04994531576291: 1, 17.026544188949103: 1, 16.970122435612016: 1,
16.969970822181146: 1, 16.93904218262121: 1, 16.758138320627946: 1, 16.719659082143377: 1,
16.55500533491189: 1, 16.498251732610044: 1, 16.443140307050975: 1, 16.39294381989571: 1,
16.337516292588248: 1, 16.336421714488893: 1, 16.268517921550963: 1, 16.267545349281345: 1,
16.158269577920766: 1, 15.97237909299115: 1, 15.960117762229503: 1, 15.90393312809645: 1,
15.87987023125909: 1, 15.876334006374266: 1, 15.87596440092019: 1, 15.854432936801071: 1,
15.840858439434493: 1, 15.834417260697576: 1, 15.702398199308591: 1, 15.65438099550754: 1,
15.577047209458971: 1, 15.575653136090624: 1, 15.538314604600062: 1, 15.508522022220944: 1,
15.394922829563889: 1, 15.389174653792539: 1, 15.381169458611957: 1, 15.357554225586155: 1,
15.170161820391598: 1, 15.046554264165882: 1, 14.983906469218642: 1, 14.947293209582407: 1,
14.892497823484895: 1, 14.851174926033064: 1, 14.810771801443488: 1, 14.76727022889918: 1,
14.766681989865171: 1, 14.733754077931286: 1, 14.65799759752674: 1, 14.586906670889341: 1,
14.560074411668083: 1, 14.452601830558025: 1, 14.348449977655964: 1, 14.32556293813643: 1,
14.23460181107868: 1, 14.22509162235754: 1, 14.17922310913186: 1, 14.1651762650674 67: 1,
14.14788199880222: 1, 14.123286255287285: 1, 14.00527734928635: 1, 13.988790344742878: 1,
13.940951556519371: 1, 13.926938451095687: 1, 13.921065052793491: 1, 13.910382236582022: 1,
13.908922736500976: 1, 13.773092229147181: 1, 13.740971583611548: 1, 13.710369292885336: 1,
13.680706601712215: 1, 13.67862142411418: 1, 13.65198671941944: 1, 13.6514917659112 58: 1,
13.634324227551852: 1, 13.587536964117513: 1, 13.585796361972204: 1, 13.558463932802738: 1,
13.547684019431792: 1, 13.544422671113814: 1, 13.470755488474289: 1, 13.467138083719101: 1,
13.452732758232704: 1, 13.451779027527609: 1, 13.424509169768514: 1, 13.415443576243238: 1,
13.41148774159879: 1, 13.406959618128681: 1, 13.375475379804035: 1, 13.351536001394287: 1,
13.298086579689866: 1, 13.290172672768922: 1, 13.19498465828456: 1, 13.175445646493193: 1,
13.161729280080607: 1, 13.151998297171328: 1, 13.14546478408489: 1, 13.138115880601742: 1,
13.084757304903395: 1, 13.074384832119362: 1, 13.059459933029881: 1, 13.022902794116444: 1,
12.901447661092535: 1, 12.888853446145784: 1, 12.888471456854292: 1, 12.88505123449083: 1,
12.88303600356633: 1, 12.862749103970911: 1, 12.858688885803163: 1, 12.799680419684067: 1,
12.752110649335762: 1, 12.739673804220882: 1, 12.645452299514819: 1, 12.612662376358 38: 1,
12.600200515399127: 1, 12.591485357295731: 1, 12.582998088679659: 1, 12.550256361190245: 1,
12.546206749235 66: 1, 12.542747184607807: 1, 12.520259669809198: 1, 12.494810859656942: 1,
12.481610085274102: 1, 12.429387600915927: 1, 12.379393981336415: 1, 12.336683630319975: 1,
12.314950829732457: 1, 12.288167381249899: 1, 12.258700699838712: 1, 12.232155126674767: 1,
12.147284797856313: 1, 12.08774380699906: 1, 12.071852154731637: 1, 12.070306459757061: 1,
12.043506412496505: 1, 12.034895691558061: 1, 12.031952954289327: 1, 12.018944360582765: 1,
12.017002388971333: 1, 11.859447033686534: 1, 11.794205368467537: 1, 11.791946124271451: 1,
11.791594976900683: 1, 11.753914027879018: 1, 11.748577744543523: 1, 11.736812435752503: 1,
11.684225061176523: 1, 11.678386044134488: 1, 11.674428663097045: 1, 11.669839024291562: 1,
11.669583309877561: 1, 11.649441873746 84: 1, 11.601937581075 99: 1, 11.597267111066694: 1,
11.505171930061602: 1, 11.464405430419346: 1, 11.454590873506977: 1, 11.452794847841695: 1,
11.449488380624361: 1, 11.437528993119003: 1, 11.427804853576488: 1, 11.415568257756545: 1,
11.408193473179782: 1, 11.397193725077242: 1, 11.315476207900726: 1, 11.313820728632411: 1,
11.310909894960236: 1, 11.302296563703477: 1, 11.28834006636902: 1, 11.25340883358174: 1,
11.241540947823687: 1, 11.234445610285707: 1, 11.181134031228455: 1, 11.170874820108645: 1,
11.164784403951195: 1, 11.163511991628276: 1, 11.163449085819574: 1, 11.160573080572854: 1,
11.158753332107494: 1, 11.124762311100358: 1, 11.117888578902068: 1, 11.104455930511003: 1,
11.071881652685962: 1, 11.058322272418353: 1, 11.023691482829 05: 1, 11.014045580388968: 1,
10.993686641239162: 1, 10.972529955923093: 1, 10.924192892021392: 1, 10.918363400520308: 1,
10.887981054958619: 1, 10.877907415648671: 1, 10.801843732408702: 1, 10.772103364415447: 1,
10.767241879392042: 1, 10.737323562547214: 1, 10.726998281402183: 1, 10.717582555194397: 1,
10.695259154292819: 1, 10.672376664100815: 1, 10.671912281691313: 1, 10.641209375120146: 1,
10.626414435951698: 1, 10.623115977827936: 1, 10.622849993122314: 1, 10.621757646404 21: 1,
10.58481503681407: 1, 10.574157557770265: 1, 10.534340908386124: 1, 10.516520422820488: 1,
10.484141698897247: 1, 10.480589055599989: 1, 10.460329219334682: 1, 10.451728089449546: 1,
10.431777707723517: 1, 10.401459333853 02: 1, 10.379823880103471: 1, 10.370045809603525: 1,
10.307001925541206: 1, 10.302292106313224: 1, 10.299480704416034: 1, 10.269343726167 55: 1,
10.25803676185813: 1, 10.220869530645844: 1, 10.206919580014981: 1, 10.174368182152916: 1,
10.168076837876077: 1, 10.166961631893544: 1, 10.155365764385134: 1, 10.153740526461089: 1,
10.150266640873644: 1, 10.136595629823763: 1, 10.133216655112376: 1, 10.124008384541215: 1,

10.150260040873044: 1, 10.150595629623765: 1, 10.135216655112576: 1, 10.124008584341215: 1,
10.105635406910729: 1, 10.103292811721973: 1, 10.103078230084826: 1, 10.085804963534995: 1,
10.083093955693547: 1, 10.07192909019787: 1, 10.053456573119918: 1, 10.051085864988346: 1,
10.046035274546593: 1, 9.99498790982893: 1, 9.952594516045611: 1, 9.932330409151586: 1,
9.917438948228565: 1, 9.907866308899457: 1, 9.901320010647085: 1, 9.887488584504345: 1,
9.884772279490013: 1, 9.835787459442702: 1, 9.834788973835446: 1, 9.827594412824423: 1,
9.776032544254702: 1, 9.759902341850024: 1, 9.743152213395009: 1, 9.732488286292291: 1,
9.729525624299997: 1, 9.703921897283502: 1, 9.70260106706044: 1, 9.681060869988137: 1,
9.674453325750104: 1, 9.619909212780797: 1, 9.605683738757996: 1, 9.591873679877523: 1,
9.55406831473125: 1, 9.526983638379978: 1, 9.519909898621993: 1, 9.508953227069917: 1,
9.453938665750117: 1, 9.44310025042109: 1, 9.429928595252415: 1, 9.425228815887854: 1,
9.423446338668446: 1, 9.403620772407749: 1, 9.359768089525335: 1, 9.355635320365204: 1,
9.340847938059222: 1, 9.33580789053087: 1, 9.332173430044977: 1, 9.332143375272887: 1,
9.302852431047583: 1, 9.284726879978535: 1, 9.238967190047225: 1, 9.238696558652913: 1,
9.236139653962196: 1, 9.231150490459335: 1, 9.22834129108939: 1, 9.221614960011252: 1,
9.217529488491623: 1, 9.213103325735855: 1, 9.18833559799277: 1, 9.161077951085254: 1,
9.136398024558945: 1, 9.11235567856614: 1, 9.098821608145052: 1, 9.090312538795567: 1,
9.08703723014767: 1, 9.084510020514417: 1, 9.075722335152143: 1, 9.063620542286428: 1,
9.056456630804878: 1, 9.051220782533612: 1, 8.994948528982865: 1, 8.976607215617571: 1,
8.975073585891701: 1, 8.97471355160394: 1, 8.96341990380725: 1, 8.949159658580147: 1,
8.944815326499661: 1, 8.91609888104398: 1, 8.901646109890713: 1, 8.89772008755449: 1,
8.883772045891172: 1, 8.86969776487416: 1, 8.864265558203801: 1, 8.861497816315314: 1,
8.854543228499276: 1, 8.848961588040824: 1, 8.84250362654262: 1, 8.842431565152516: 1,
8.832144804756451: 1, 8.829388731617609: 1, 8.821511314203798: 1, 8.81984647640449: 1,
8.812082688462638: 1, 8.809447194392146: 1, 8.802769708742401: 1, 8.80203890169436: 1,
8.801741836710582: 1, 8.792604753681614: 1, 8.788592416938679: 1, 8.760894847692375: 1,
8.759204924222386: 1, 8.758585118225632: 1, 8.751064183160633: 1, 8.729757768290785: 1,
8.706904572060434: 1, 8.696912695446802: 1, 8.674411554587051: 1, 8.668368301611826: 1,
8.660573130135072: 1, 8.642270262464674: 1, 8.633237300344055: 1, 8.62989695305115: 1,
8.615281919886952: 1, 8.589261239053187: 1, 8.566176842257095: 1, 8.565315368685825: 1,
8.560677608344124: 1, 8.528167787507472: 1, 8.515466397238564: 1, 8.494340996129166: 1,
8.489659016761399: 1, 8.438580486616862: 1, 8.42713334467582: 1, 8.405368133606437: 1,
8.403776198797365: 1, 8.396049961813212: 1, 8.39425203202048: 1, 8.38752484250685: 1,
8.370836150191366: 1, 8.356551872473357: 1, 8.318977421354797: 1, 8.301509074746978: 1,
8.288068118649523: 1, 8.251634996434442: 1, 8.230548596041256: 1, 8.227380604183102: 1,
8.193031144358562: 1, 8.181471667724093: 1, 8.179000203237354: 1, 8.172788493464497: 1,
8.157448963766: 1, 8.15696975510188: 1, 8.13185515971308: 1, 8.131163626174079: 1,
8.126890056809485: 1, 8.096235203235572: 1, 8.085098867237372: 1, 8.07203439753088: 1,
8.05704047059476: 1, 8.049010559335253: 1, 8.031005749378167: 1, 8.01714489331686: 1,
7.978889387603512: 1, 7.974478078584135: 1, 7.92263494271559: 1, 7.9224000503732395: 1,
7.880777455076518: 1, 7.867654220937664: 1, 7.8397825847945155: 1, 7.8393922282190855: 1,
7.8358522113026785: 1, 7.831663936095581: 1, 7.826038807577384: 1, 7.8124582062842585: 1,
7.809925586554973: 1, 7.805203361962002: 1, 7.793781283824739: 1, 7.781791663737022: 1,
7.767771338536488: 1, 7.756046399392955: 1, 7.750740748257593: 1, 7.739264826351866: 1,
7.729318795437778: 1, 7.717926927039595: 1, 7.706182184599623: 1, 7.688480259605548: 1,
7.6825970453109225: 1, 7.679854628429355: 1, 7.671225313291656: 1, 7.658582547670097: 1, 7.65674151
1393956: 1, 7.656664025676743: 1, 7.654306995535531: 1, 7.642944227884216: 1, 7.61908045946762: 1,
7.597657829410283: 1, 7.596147580863695: 1, 7.594701113528473: 1, 7.582972558421743: 1,
7.569804790120054: 1, 7.562480574686681: 1, 7.555941786697716: 1, 7.554371225123041: 1,
7.546939356278993: 1, 7.529161698780092: 1, 7.480780386931148: 1, 7.458858849075524: 1,
7.457534247586221: 1, 7.446720317207281: 1, 7.442713012804037: 1, 7.429614173956446: 1,
7.428640798596244: 1, 7.4185390324727605: 1, 7.410556121585735: 1, 7.409562518616227: 1,
7.383953781872309: 1, 7.381513352944903: 1, 7.378153826290574: 1, 7.337792358746611: 1,
7.337286814031891: 1, 7.323651976463674: 1, 7.32358764881274: 1, 7.320753760079386: 1,
7.317529713927735: 1, 7.315178116494996: 1, 7.301695301643981: 1, 7.27861463085962: 1,
7.278230182487505: 1, 7.277076566937836: 1, 7.275704827560757: 1, 7.251349133075176: 1,
7.2409143644726806: 1, 7.214674869586268: 1, 7.182314652567965: 1, 7.1644889018321765: 1,
7.154071689321237: 1, 7.14925710189716: 1, 7.130061412586181: 1, 7.1234973004865685: 1,
7.098877378242539: 1, 7.088919761999456: 1, 7.0868436556456205: 1, 7.063999937315306: 1,
7.058510400342943: 1, 7.047373619049396: 1, 7.024105143005853: 1, 7.024086118223103: 1,
7.023274631098147: 1, 7.0213277053841: 1, 7.004520399657462: 1, 6.9931034215265235: 1,
6.9904025905814935: 1, 6.983119404419805: 1, 6.97490220290545: 1, 6.9672852238883465: 1,
6.9663478777340355: 1, 6.966129462646844: 1, 6.962856070006974: 1, 6.957364580095332: 1, 6.94448234
8972527: 1, 6.9318125565483895: 1, 6.928322314944597: 1, 6.922876254968707: 1, 6.918807067128801:
1, 6.91506449637807: 1, 6.883111501484238: 1, 6.864686336744598: 1, 6.860348976962909: 1,
6.847234876055409: 1, 6.839457673857983: 1, 6.825219689153776: 1, 6.790431501637338: 1,
6.783594737648612: 1, 6.7796103402804535: 1, 6.774581752508771: 1, 6.77248773314576: 1,
6.769209845107229: 1, 6.765461520060945: 1, 6.764275743124607: 1, 6.753901156577996: 1,
6.75323146718007: 1, 6.7515252410533915: 1, 6.747284780141234: 1, 6.741640437061254: 1,
6.729224621536025: 1, 6.727757100980392: 1, 6.721092394517036: 1, 6.718812642556657: 1,
6.718423516668145: 1, 6.715390022708349: 1, 6.71233487250277: 1, 6.693780263897153: 1,
6.692360101333388: 1, 6.688767344681018: 1, 6.664794671718396: 1, 6.644854142745772: 1,
6.642677235221358: 1, 6.641464894614375: 1, 6.629070880792718: 1, 6.627637888667957: 1,
6.603809830239952: 1, 6.597899941463072: 1, 6.584678962073549: 1, 6.574581142065724: 1,
6.571526483790022: 1, 6.570857790254185: 1, 6.556726312849586: 1, 6.550093014127917: 1,
6.549292766526499: 1, 6.543125379754719: 1, 6.542819850945542: 1, 6.541129228567047: 1,
6.53459290508834: 1, 6.534462904712606: 1, 6.534018697554591: 1, 6.5276708281105185: 1,
6.520084346154751: 1, 6.51586254474975: 1, 6.5075077012320275: 1, 6.491407910351515: 1

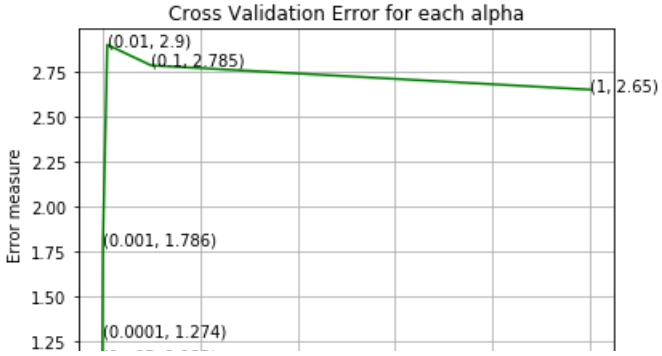
6.4796050501095985: 1, 6.4690397705236675: 1, 6.46680176664895: 1, 6.465153615035423: 1,
6.4631022285887205: 1, 6.453078064319994: 1, 6.4450311120126145: 1, 6.428132799610293: 1,
6.423441941979651: 1, 6.401299756504 78: 1, 6.383959592047209: 1, 6.378899370833185: 1,
6.376937356904242: 1, 6.3655138926714585: 1, 6.354539055480764: 1, 6.353761178328422: 1,
6.345767478653709: 1, 6.306856192812307: 1, 6.291651767416033: 1, 6.288244560123476: 1,
6.2795818409383735: 1, 6.278621811973729: 1, 6.258718910710207: 1, 6.244406716847924: 1, 6.24276604
8687967: 1, 6.24276011057421: 1, 6.235397070396116: 1, 6.229256936841064: 1, 6.220505153596666: 1,
6.196768817411081: 1, 6.1966717572774055: 1, 6.179918730683418: 1, 6.172575848800656: 1,
6.17207611813689: 1, 6.167506225002532: 1, 6.164468159324432: 1, 6.154684442113105: 1,
6.151130148202465: 1, 6.148784369273708: 1, 6.144941085972318: 1, 6.127453515748124: 1,
6.123191457731809: 1, 6.109791979144023: 1, 6.100349443910714: 1, 6.097727785811686: 1,
6.094396551701862: 1, 6.093243875727278: 1, 6.09097518059162: 1, 6.0879613591552335: 1,
6.085294871149201: 1, 6.063056374106573: 1, 6.056876581754889: 1, 6.036073555303236: 1,
6.034067618906684: 1, 6.031640590639793: 1, 6.028452109623083: 1, 6.017601661691343: 1,
6.012912789111521: 1, 6.0030447766558686: 1, 5.984291266027185: 1, 5.983108478092903: 1,
5.980771682137797: 1, 5.979626401446103: 1, 5.978826066675954: 1, 5.97431398825576: 1,
5.974290529844997: 1, 5.967107922954578: 1, 5.963813083411246: 1, 5.95841894328705: 1,
5.942518377333757: 1, 5.9418692890307: 1, 5.9404668555504904: 1, 5.94013861389795 75: 1,
5.9383086055960455: 1, 5.930027619602237: 1, 5.9099091715974685: 1, 5.905435106593141: 1,
5.904032854427502: 1, 5.899174277707609: 1, 5.896444007530275: 1, 5.891535894139586: 1,
5.8882237931988675: 1, 5.887672415856108: 1, 5.877450990362074: 1, 5.866406199620235: 1, 5.86135351
5910242: 1, 5.851501010867176: 1, 5.842317993963618: 1, 5.838577139613703: 1, 5.837147615774579: 1,
5.836475318776488: 1, 5.832752353390367: 1, 5.814375003636273: 1, 5.8131441037065 61: 1,
5.7966122042046155: 1, 5.789326053540027: 1, 5.788767473615105: 1, 5.776870931067305: 1, 5.77629208
2270499: 1, 5.761898860584853: 1, 5.7577228286755915: 1, 5.747013572700927: 1, 5.746229963922788:
1, 5.740920870893917: 1, 5.734179624418339: 1, 5.7256389687458 52: 1, 5.7255313147879: 1,
5.71869679320882: 1, 5.716697527964719: 1, 5.707635838737522: 1, 5.698537804352049: 1,
5.67406043014988: 1, 5.674012343692744: 1, 5.668845995763953: 1, 5.658409230960243: 1,
5.6531768245789875: 1, 5.64744687757556: 1, 5.647280184883013: 1, 5.637476669113184: 1,
5.633134465816731: 1, 5.627390246692286: 1, 5.626127922038992: 1, 5.619550145137611: 1,
5.610000964494649: 1, 5.606417114476137: 1, 5.602031895021569: 1, 5.594857348257976: 1,
5.591910385284526: 1, 5.574808723448591: 1, 5.573050020404478: 1, 5.568531957815523: 1,
5.562620313053115: 1, 5.557274885173739: 1, 5.557015615609928: 1, 5.550376334704487: 1,
5.547627960069936: 1, 5.543950367626225: 1, 5.53866851100249: 1, 5.536099931906267: 1,
5.523347745422631: 1, 5.521478369599746: 1, 5.50545458021 68: 1, 5.504658221854936: 1,
5.503522911384304: 1, 5.500650954941393: 1, 5.488979418131516: 1, 5.481460544941642: 1,
5.47000357787808: 1, 5.468692847542458: 1, 5.464615349385 65: 1, 5.461728980663659: 1,
5.45649865940514: 1, 5.450305330537092: 1, 5.436218095151408: 1, 5.436205446761638: 1,
5.430421035719382: 1, 5.427885034585383: 1, 5.42771125185 95: 1, 5.420025598372765: 1,
5.415388973549299: 1, 5.407361449463664: 1, 5.406921078713909: 1, 5.396602443809952: 1,
5.393045854310949: 1, 5.384583133928549: 1, 5.373987032324353: 1, 5.372009863921235: 1,
5.368653953307773: 1, 5.366088544540213: 1, 5.365983127554974: 1, 5.353631283902732: 1,
5.351746572291636: 1, 5.345114193208501: 1, 5.343595960837935: 1, 5.342767479297 62: 1,
5.342506862838823: 1, 5.3367575662647235: 1, 5.31398282137783: 1, 5.306501709312489: 1,
5.304475422038575: 1, 5.297509479511016: 1, 5.248415939642299: 1, 5.229479465959776: 1,
5.21593373048725: 1, 5.195218689591472: 1, 5.186673573088557: 1, 5.173475391544212: 1,
5.162554812774275: 1, 5.162147855931324: 1, 5.158623172966471: 1, 5.154015528507104: 1,
5.151982369722278: 1, 5.151737986418728: 1, 5.151378505286954: 1, 5.138009156433096: 1,
5.130632968415511: 1, 5.128560418079184: 1, 5.103213693608334: 1, 5.100349434523588: 1,
5.09827230205886: 1, 5.080069185593164: 1, 5.076413258207445: 1, 5.074819972416119: 1,
5.073286531932867: 1, 5.060529358362643: 1, 5.059465665825107: 1, 5.052139604149107: 1,
5.035424520659729: 1, 4.992911561769389: 1, 4.974591005076599: 1, 4.968756369048756: 1,
4.9610205217186225: 1, 4.951127121074852: 1, 4.944324419374177: 1, 4.927489793109642: 1, 4.90385971
0921952: 1, 4.900797826220733: 1, 4.898702456189159: 1, 4.890213987027324: 1, 4.882732949962578: 1,
4.877018536863204: 1, 4.875094056504045: 1, 4.8743429195869545: 1, 4.859316415654721: 1,
4.858216319464034: 1, 4.856440537541518: 1, 4.853472238107031: 1, 4.8490001601933646: 1,
4.845796929582605: 1, 4.835574158743906: 1, 4.8256387006980015: 1, 4.816191610687267: 1,
4.813377299061662: 1, 4.811868239368928: 1, 4.8100494184159945: 1, 4.802882815954293: 1,
4.793873094913339: 1, 4.7715155964634315: 1, 4.755671326592746: 1, 4.755657338077542: 1,
4.698885074847695: 1, 4.695789396587489: 1, 4.687512083463241: 1, 4.677422469398158: 1,
4.675723824914886: 1, 4.670850257680993: 1, 4.665464950758632: 1, 4.65687124830455: 1,
4.643744937702463: 1, 4.625140163847068: 1, 4.623886175983986: 1, 4.613414791802 89: 1,
4.610751093176521: 1, 4.587810911024009: 1, 4.585787192178812: 1, 4.570228241888739: 1,
4.540902264883619: 1, 4.540293060303336: 1, 4.516919694534822: 1, 4.5137797308416445: 1,
4.47961389478384: 1, 4.444748240566349: 1, 4.439007738035377: 1, 4.3591423764349235: 1,
4.355043858592118: 1, 4.347849926629424: 1, 4.3383140628100385: 1, 4.319637149249234: 1,
4.307542232034905: 1, 4.285349171274039: 1, 4.162304955320652: 1, 4.144199908618865: 1})

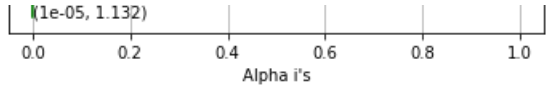
In [57]:

```
# 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: 1.1322445686675882
For values of alpha =  0.0001 The log loss is: 1.2740045813223688
For values of alpha =  0.001 The log loss is: 1.7856421231791488
For values of alpha =  0.01 The log loss is: 2.9002351790107435
For values of alpha =  0.1 The log loss is: 2.7848497769263836
For values of alpha =  1 The log loss is: 2.649973496993815
```






```
For values of best alpha =  1e-05 The train log loss is: 0.7552178887394458
For values of best alpha =  1e-05 The cross validation log loss is: 1.1322445686675882
For values of best alpha =  1e-05 The test log loss is: 1.107267006768426
```

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

Ans. Yes, it seems like!

In [58]:

```
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 [59]:

```
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")
```

```
3.361 % of word of test data appeared in train data
3.874 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [60]:

```
#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 [61]:

```
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 [72]:

```
# 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 = TfidfVectorizer(min_df=3,max_features=1000,ngram_range=(1,2))

    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 [64]:

```
# 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 [65]:

```
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, 3196)
(number of data points * number of features) in test data =  (665, 3196)
(number of data points * number of features) in cross validation data = (532, 3196)
```

In [66]:

```
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

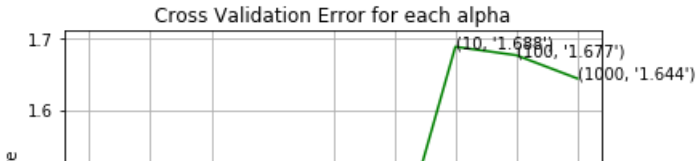
4.1.1.1. Hyper parameter tuning

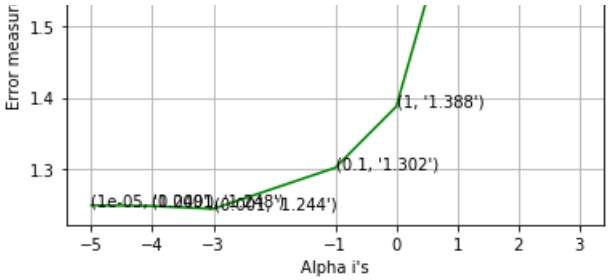
In [67]:

```
# 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()
```

```
# 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 : 1.2493912269628704
for alpha = 0.0001
Log Loss : 1.2483749352779177
for alpha = 0.001
Log Loss : 1.244373532401428
for alpha = 0.1
Log Loss : 1.3020229565846857
for alpha = 1
Log Loss : 1.38766424783384
for alpha = 10
Log Loss : 1.6884254298779762
for alpha = 100
Log Loss : 1.6765128742412991
for alpha = 1000
Log Loss : 1.6439425275448831
```





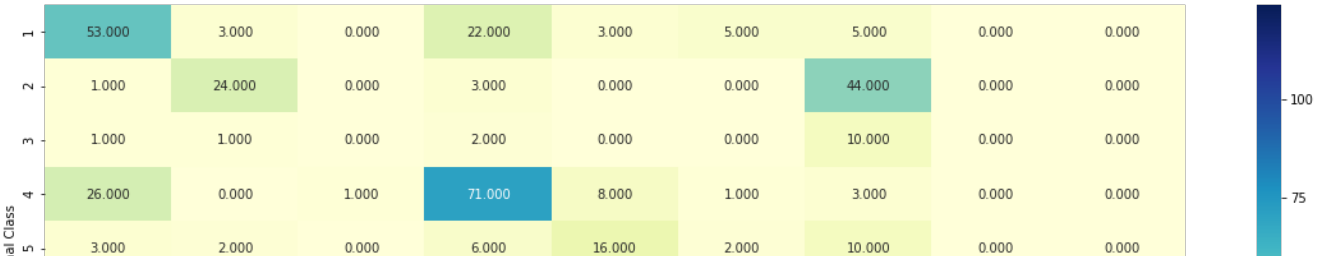

```
For values of best alpha =  0.001 The train log loss is: 0.49294618752910274
For values of best alpha =  0.001 The cross validation log loss is: 1.244373532401428
For values of best alpha =  0.001 The test log loss is: 1.1519047183663926
```

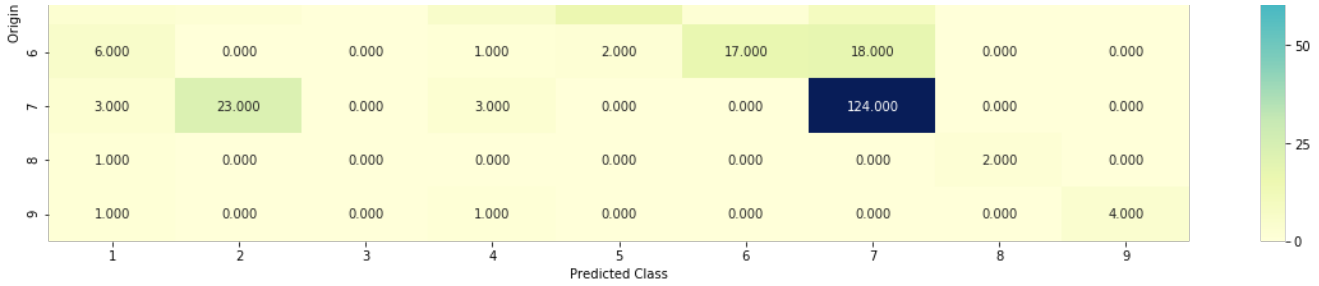
4.1.1.2. Testing the model with best hyper paramters

In [68]:

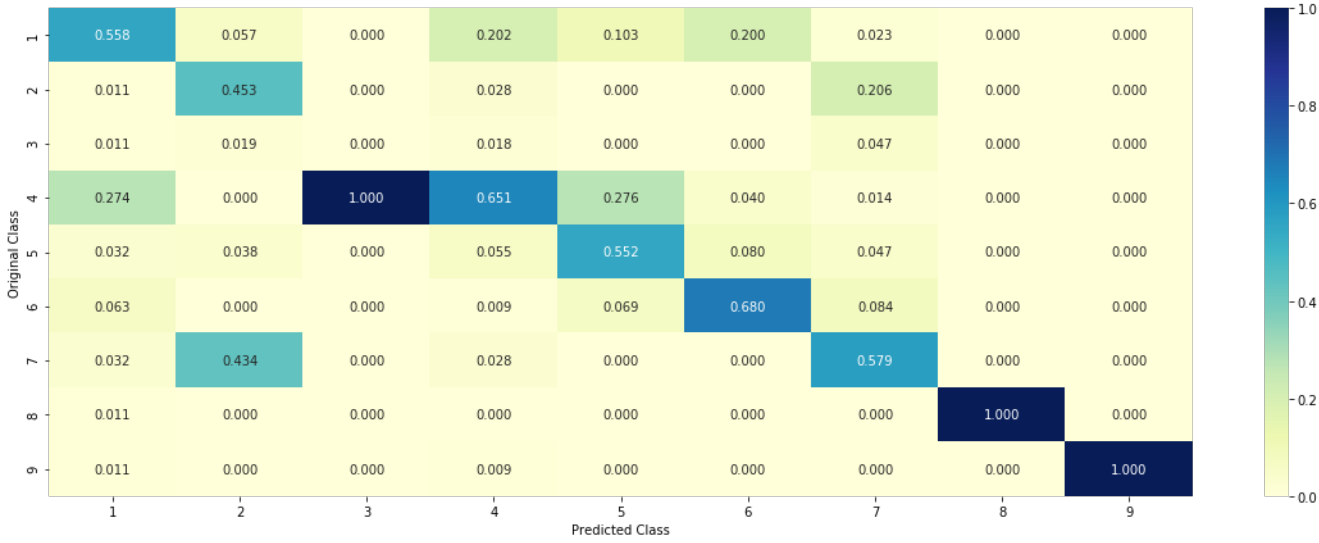
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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 : 1.244373532401428
Number of missclassified point : 0.41541353383458646
-------------------- Confusion matrix --------------------
```

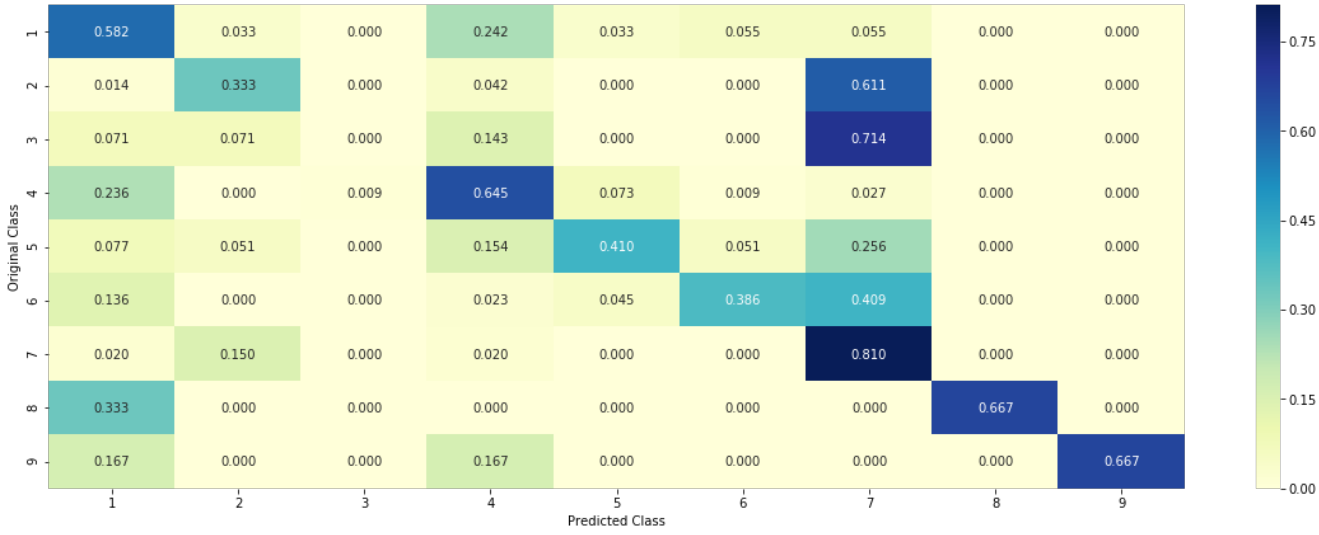





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.1.1.3. Feature Importance, Correctly classified point

In [73]:

```
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: [[0.0545 0.0382 0.0118 0.0624 0.0312 0.0304 0.7663 0.0023 0.0027]]
Actual Class : 2
--------------------------------------------------
15 Text feature [alone] present in test data point [True]
17 Text feature [individual] present in test data point [True]
18 Text feature [assay] present in test data point [True]
19 Text feature [allele] present in test data point [True]
22 Text feature [as] present in test data point [True]
27 Text feature [overall] present in test data point [True]
28 Text feature [when] present in test data point [True]
29 Text feature [were] present in test data point [True]
31 Text feature [through] present in test data point [True]
35 Text feature [part] present in test data point [True]
38 Text feature [three] present in test data point [True]
39 Text feature [its] present in test data point [True]
42 Text feature [culture] present in test data point [True]
47 Text feature [because] present in test data point [True]
50 Text feature [although] present in test data point [True]
56 Text feature [pathway] present in test data point [True]
58 Text feature [least] present in test data point [True]
64 Text feature [while] present in test data point [True]
65 Text feature [impact] present in test data point [True]
68 Text feature [normal] present in test data point [True]
73 Text feature [genes] present in test data point [True]
82 Text feature [those] present in test data point [True]
83 Text feature [alleles] present in test data point [True]
85 Text feature [molecular] present in test data point [True]
89 Text feature [blot] present in test data point [True]
91 Text feature [less] present in test data point [True]
94 Text feature [number] present in test data point [True]
95 Text feature [known] present in test data point [True]
96 Text feature [pathways] present in test data point [True]
98 Text feature [no] present in test data point [True]
Out of the top  100  features  30 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [75]:

```
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 : 5
Predicted Class Probabilities: [[0.1251 0.0553 0.0191 0.1051 0.4878 0.0834 0.1162 0.0038 0.0043]]
Actual Class : 5
--------------------------------------------------
8 Text feature [was] present in test data point [True]
21 Text feature [should] present in test data point [True]
23 Text feature [selected] present in test data point [True]
32 Text feature [than] present in test data point [True]
33 Text feature [small] present in test data point [True]
39 Text feature [they] present in test data point [True]
43 Text feature [examined] present in test data point [True]
48 Text feature [although] present in test data point [True]
49 Text feature [any] present in test data point [True]
52 Text feature [through] present in test data point [True]
58 Text feature [mutations] present in test data point [True]
60 Text feature [impact] present in test data point [True]
61 Text feature [less] present in test data point [True]
62 Text feature [three] present in test data point [True]
```

```
73 Text feature [conserved] present in test data point [True]
78 Text feature [while] present in test data point [True]
84 Text feature [30] present in test data point [True]
85 Text feature [with] present in test data point [True]
86 Text feature [database] present in test data point [True]
87 Text feature [assay] present in test data point [True]
88 Text feature [known] present in test data point [True]
93 Text feature [individual] present in test data point [True]
Out of the top  100  features  22 are present in query point
```

4.2. K Nearest Neighbour Classification

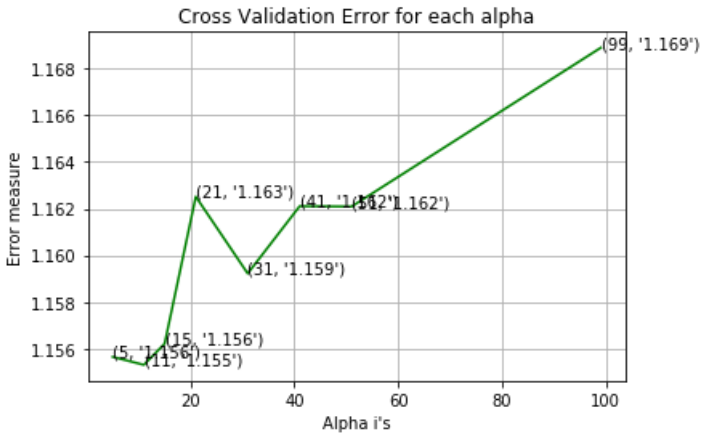
4.2.1. Hyper parameter tuning

In [76]:

```
# 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 : 1.1556470126531952
for alpha = 11
Log Loss : 1.1553044206997856
for alpha = 15
Log Loss : 1.1562136536461665
for alpha = 21
Log Loss : 1.1625139236327486
for alpha = 31
Log Loss : 1.1592188269566095
for alpha = 41
Log Loss : 1.1621035266306663
for alpha = 51
Log Loss : 1.1620874612697678
for alpha = 99
Log Loss : 1.1688952018648577
```




```
For values of best alpha =  11 The train log loss is: 0.5840797905688578
For values of best alpha =  11 The cross validation log loss is: 1.1553044206997856
For values of best alpha =  11 The test log loss is: 1.0996138703742573
```

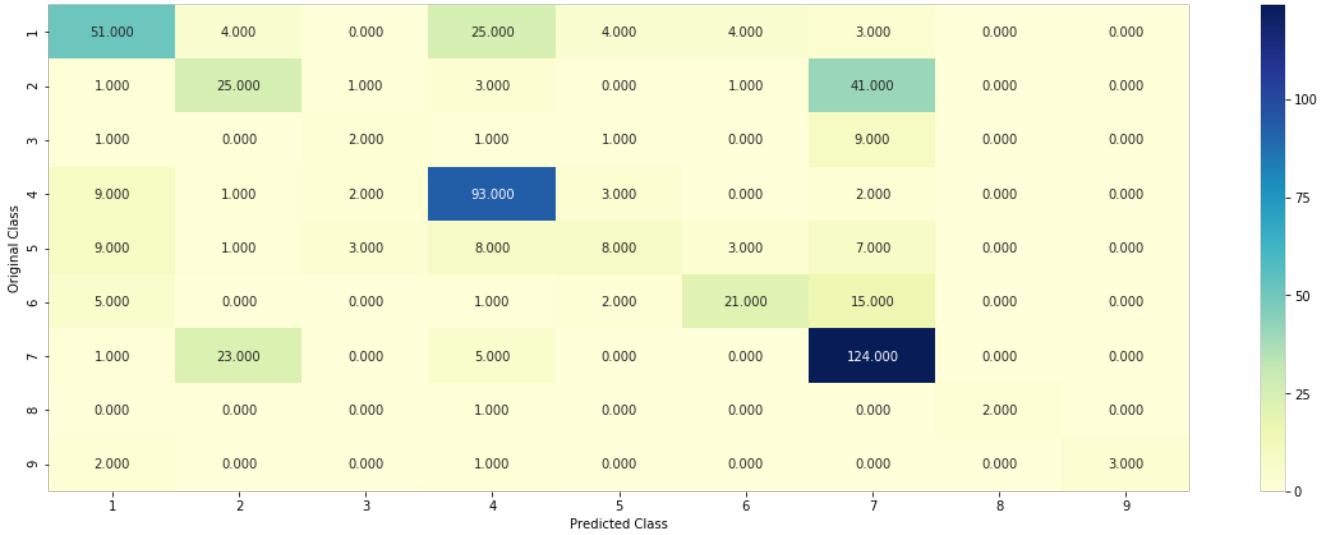
4.2.2. Testing the model with best hyper paramters

In [77]:

```
# 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/
#-------------------------------------
```

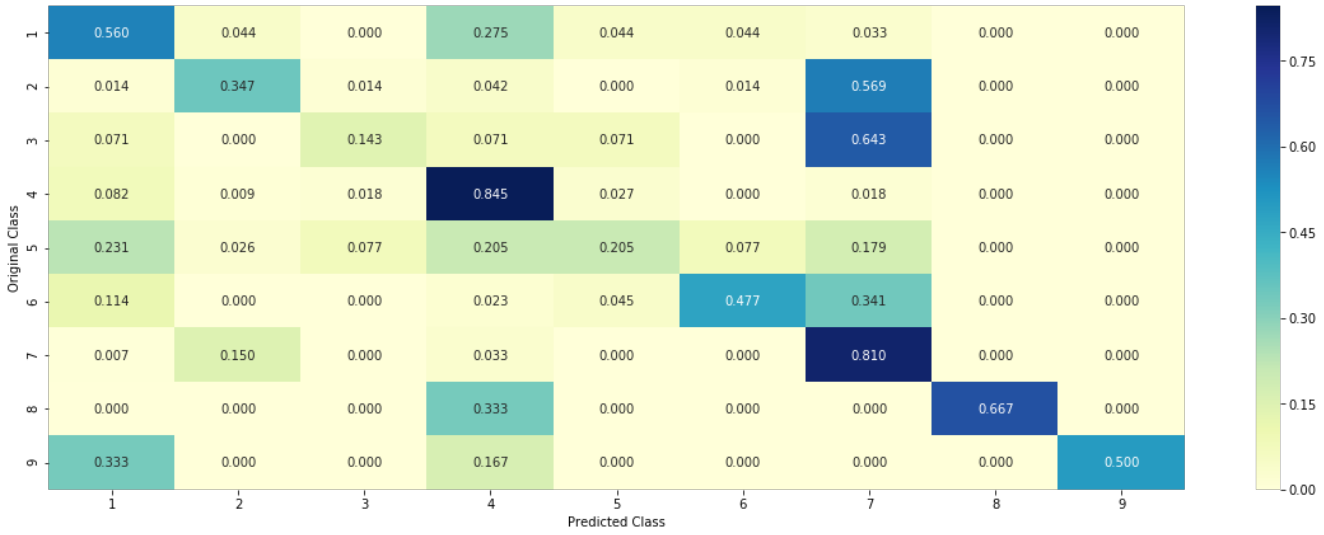
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.1553044206997856
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```




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


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



4.2.3.Sample Query point -1

In [78]:

```
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 : 4
Actual Class : 2
The  11  nearest neighbours of the test points belongs to classes [2 7 7 7 7 7 7 7 7 7 7]
Fequency of nearest points : Counter({7: 10, 2: 1})
```

4.2.4. Sample Query Point-2

In [79]:

```
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 : 1
Actual Class : 5
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [1 1 5
5 1 6 5 6 4 1 1]
Fequency of nearest points : Counter({1: 5, 5: 3, 6: 2, 4: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

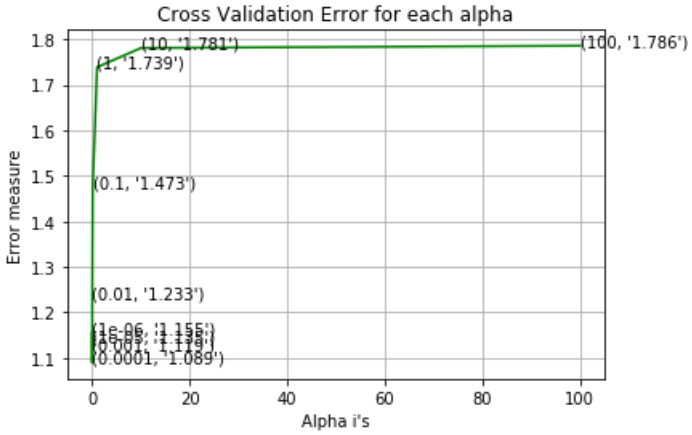
In [80]:

```
# 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
```

```
# 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 : 1.155122819746155
for alpha = 1e-05
Log Loss : 1.134872838748283
for alpha = 0.0001
Log Loss : 1.089122687167706
```

```
for alpha = 0.001
Log Loss : 1.1185083655889243
for alpha = 0.01
Log Loss : 1.232625722851298
for alpha = 0.1
Log Loss : 1.4727055958029265
for alpha = 1
Log Loss : 1.7385607480778749
for alpha = 10
Log Loss : 1.7807230737087678
for alpha = 100
Log Loss : 1.7856851450127438
```



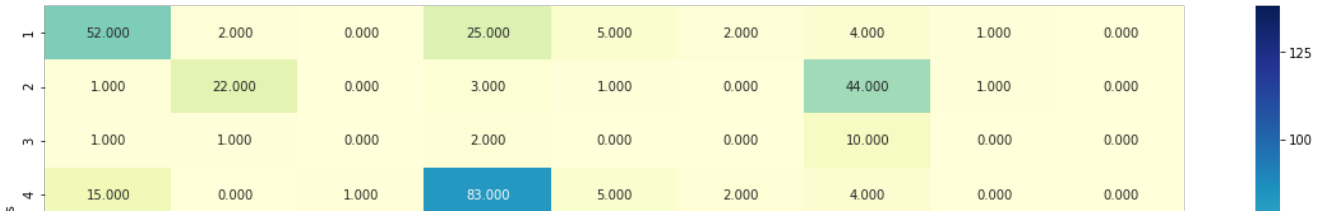

```
For values of best alpha =  0.0001 The train log loss is: 0.4179357285871196
For values of best alpha =  0.0001 The cross validation log loss is: 1.089122687167706
For values of best alpha =  0.0001 The test log loss is: 1.0072177697071847
```

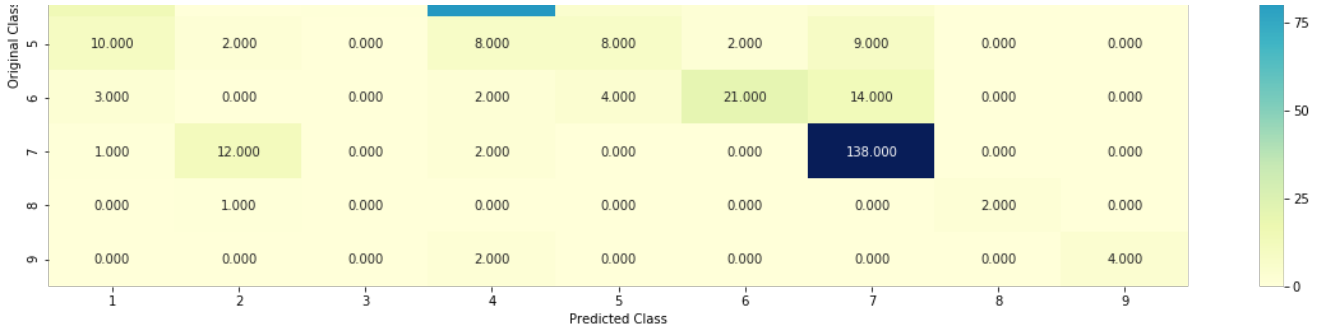
4.3.1.2. Testing the model with best hyper paramters

In [81]:

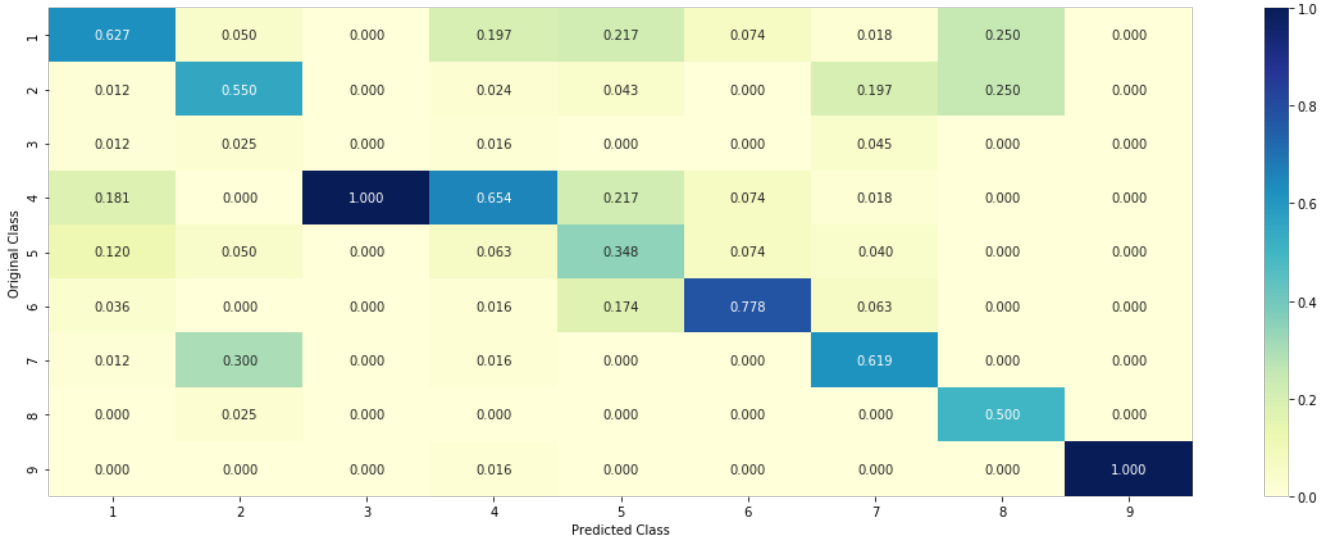
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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 : 1.089122687167706
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```

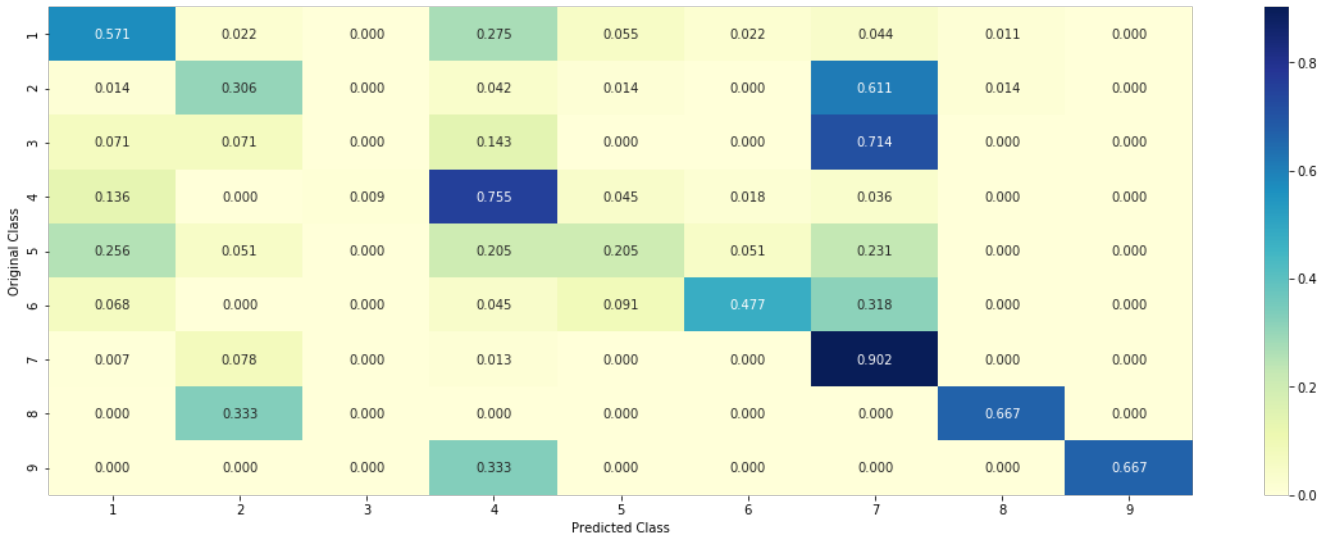





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.3.1.3. Feature Importance

In [82]:

```
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 [83]:

```
# 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: [[0.0149 0.0769 0.0044 0.0301 0.0069 0.0009 0.8622 0.0012 0.0025]]
Actual Class : 2
--------------------------------------------------
15 Text feature [alone] present in test data point [True]
19 Text feature [allele] present in test data point [True]
45 Text feature [type] present in test data point [True]
48 Text feature [by] present in test data point [True]
92 Text feature [it] present in test data point [True]
130 Text feature [complete] present in test data point [True]
136 Text feature [all] present in test data point [True]
198 Text feature [lower] present in test data point [True]
241 Text feature [buffer] present in test data point [True]
242 Text feature [higher] present in test data point [True]
260 Text feature [alleles] present in test data point [True]
261 Text feature [its] present in test data point [True]
263 Text feature [one] present in test data point [True]
306 Text feature [min] present in test data point [True]
308 Text feature [change] present in test data point [True]
333 Text feature [other] present in test data point [True]
335 Text feature [present] present in test data point [True]
358 Text feature [proteins] present in test data point [True]
371 Text feature [receptors] present in test data point [True]
379 Text feature [formation] present in test data point [True]
386 Text feature [constructs] present in test data point [True]
404 Text feature [in] present in test data point [True]
408 Text feature [amino] present in test data point [True]
412 Text feature [further] present in test data point [True]
439 Text feature [substitutions] present in test data point [True]
455 Text feature [forms] present in test data point [True]
465 Text feature [expression] present in test data point [True]
495 Text feature [lines] present in test data point [True]
498 Text feature [because] present in test data point [True]
Out of the top  500  features  29 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [84]:

```
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 : 1
Predicted Class Probabilities: [[0.5193 0.0033 0.0096 0.1178 0.2835 0.0634 0.001  0.0009 0.0012]]
Actual Class : 5
--------------------------------------------------
176 Text feature [did] present in test data point [True]
189 Text feature [can] present in test data point [True]
213 Text feature [phenotype] present in test data point [True]
268 Text feature [significant] present in test data point [True]
293 Text feature [classification] present in test data point [True]
294 Text feature [deleterious] present in test data point [True]
306 Text feature [positive] present in test data point [True]
312 Text feature [methods] present in test data point [True]
319 Text feature [similar] present in test data point [True]
321 Text feature [whether] present in test data point [True]
328 Text feature [strong] present in test data point [True]
331 Text feature [independent] present in test data point [True]
332 Text feature [patients] present in test data point [True]
335 Text feature [identify] present in test data point [True]
339 Text feature [above] present in test data point [True]
340 Text feature [interactions] present in test data point [True]
347 Text feature [patient] present in test data point [True]
362 Text feature [we] present in test data point [True]
374 Text feature [would] present in test data point [True]
385 Text feature [indicated] present in test data point [True]
387 Text feature [including] present in test data point [True]
399 Text feature [generated] present in test data point [True]
401 Text feature [some] present in test data point [True]
405 Text feature [alterations] present in test data point [True]
407 Text feature [high] present in test data point [True]
413 Text feature [cases] present in test data point [True]
416 Text feature [four] present in test data point [True]
421 Text feature [tested] present in test data point [True]
424 Text feature [side] present in test data point [True]
428 Text feature [following] present in test data point [True]
446 Text feature [gene] present in test data point [True]
448 Text feature [such] present in test data point [True]
453 Text feature [yeast] present in test data point [True]
468 Text feature [domain] present in test data point [True]
475 Text feature [obtained] present in test data point [True]
481 Text feature [protein] present in test data point [True]
482 Text feature [have] present in test data point [True]
494 Text feature [showed] present in test data point [True]
495 Text feature [while] present in test data point [True]
Out of the top  500  features  39 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

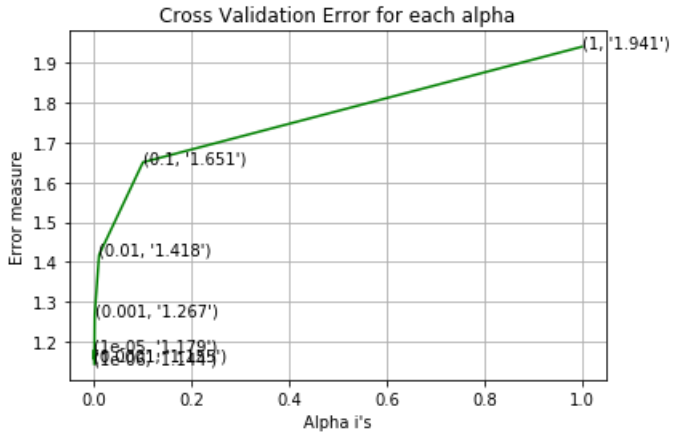
In [85]:

```
# 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-intuition-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_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 alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.144010414129801
for alpha = 1e-05
Log Loss : 1.1790307302461671
for alpha = 0.0001
Log Loss : 1.155044525711724
for alpha = 0.001
Log Loss : 1.2673747745020074
for alpha = 0.01
Log Loss : 1.4178518877096593
```

```
for alpha = 0.1
Log Loss : 1.6505407083545938
for alpha = 1
Log Loss : 1.9407926628443106
```



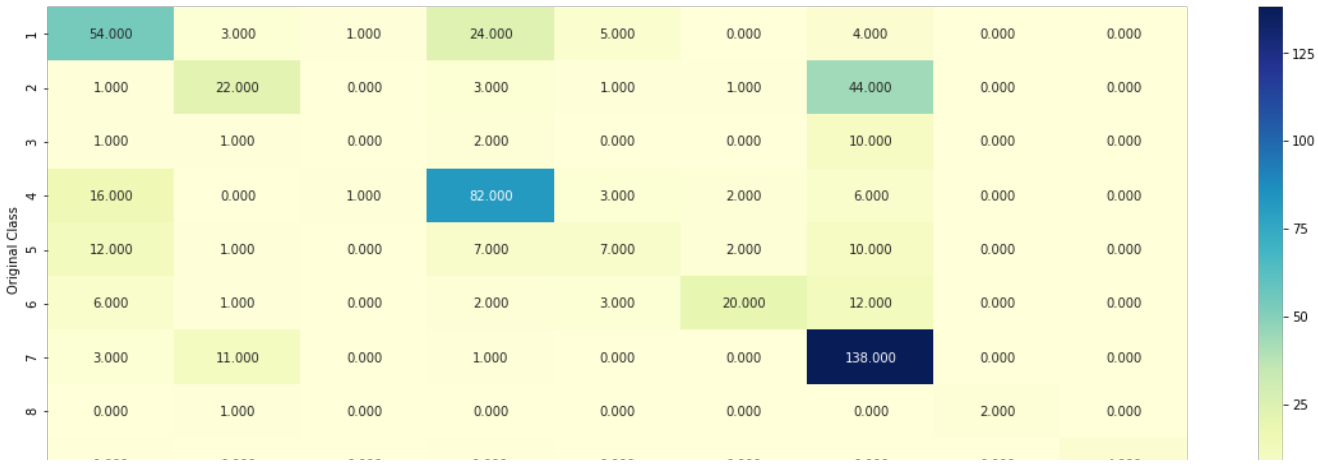

```
For values of best alpha =  1e-06 The train log loss is: 0.508970514682997
For values of best alpha =  1e-06 The cross validation log loss is: 1.144010414129801
For values of best alpha =  1e-06 The test log loss is: 1.126566442986024
```

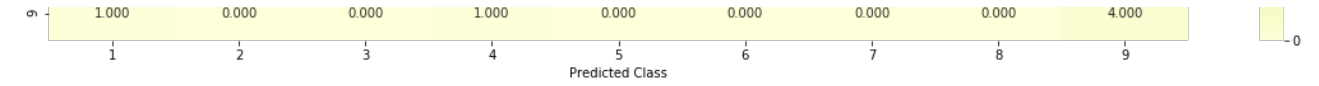
4.3.2.2. Testing model with best hyper parameters

In [86]:

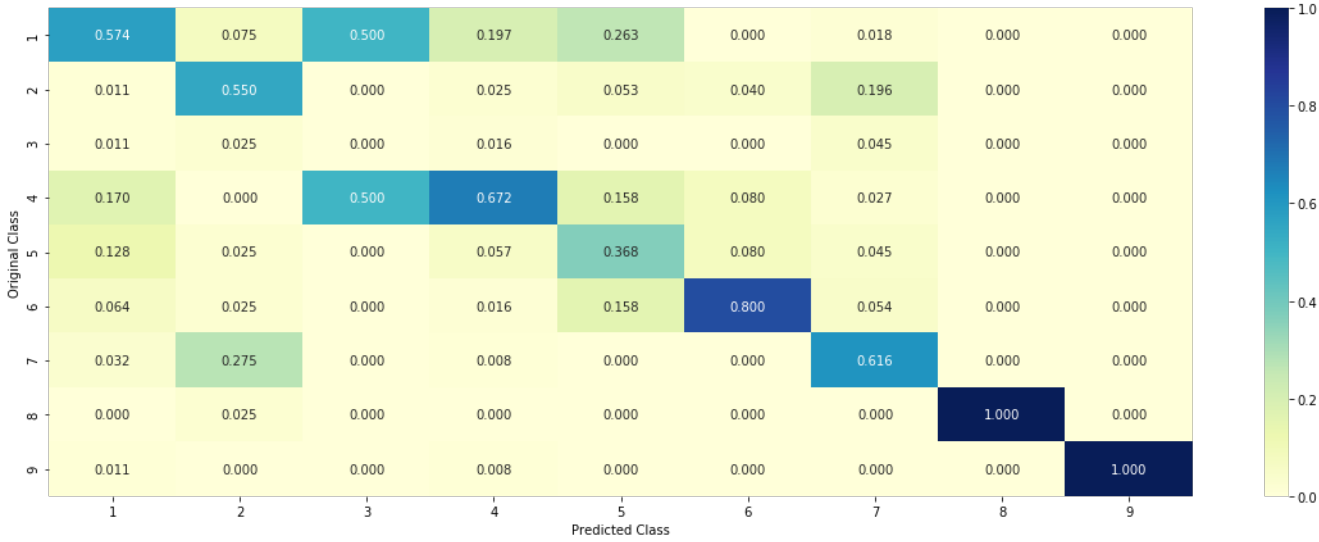
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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 : 1.144010414129801
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```

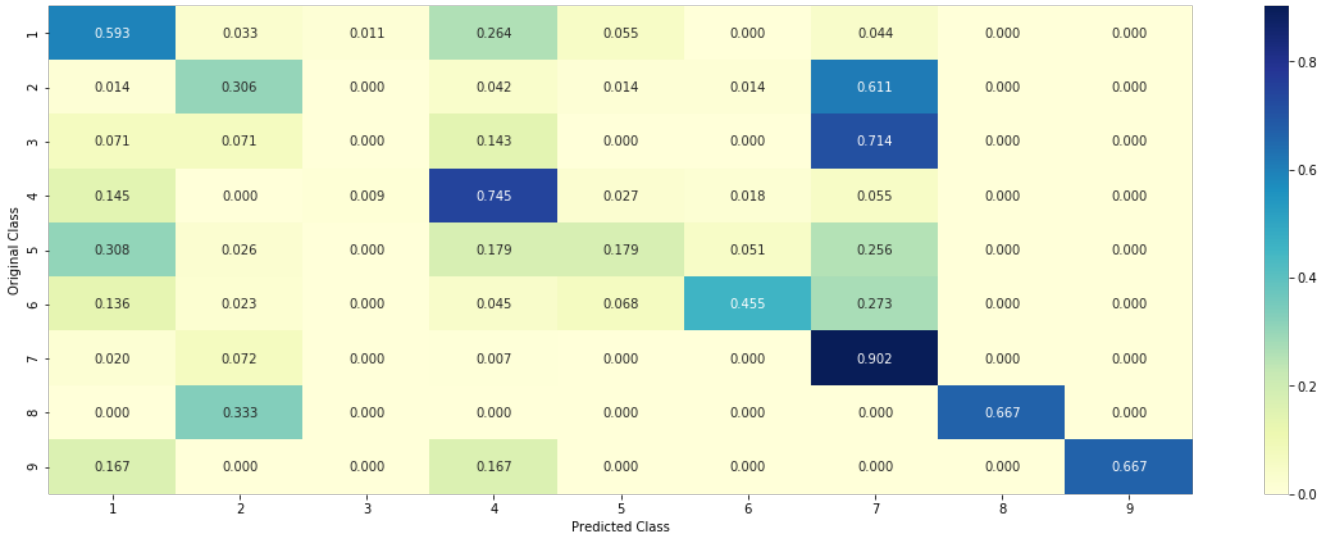





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.3.2.3. Feature Importance, Correctly Classified point

In [87]:

```
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: [[1.030e-02 1.112e-01 1.610e-02 1.185e-01 1.590e-02 5.000e-04 7.244
e-01
```

```
e-01
  5.000e-04 2.700e-03]]
Actual Class : 2
--------------------------------------------------
49 Text feature [it] present in test data point [True]
52 Text feature [by] present in test data point [True]
71 Text feature [type] present in test data point [True]
72 Text feature [higher] present in test data point [True]
75 Text feature [other] present in test data point [True]
86 Text feature [alone] present in test data point [True]
104 Text feature [amino] present in test data point [True]
353 Text feature [change] present in test data point [True]
365 Text feature [lower] present in test data point [True]
386 Text feature [its] present in test data point [True]
402 Text feature [because] present in test data point [True]
405 Text feature [proteins] present in test data point [True]
410 Text feature [forms] present in test data point [True]
411 Text feature [min] present in test data point [True]
413 Text feature [complete] present in test data point [True]
415 Text feature [in] present in test data point [True]
417 Text feature [allele] present in test data point [True]
431 Text feature [substitutions] present in test data point [True]
433 Text feature [all] present in test data point [True]
435 Text feature [constructs] present in test data point [True]
436 Text feature [formation] present in test data point [True]
444 Text feature [one] present in test data point [True]
447 Text feature [suggest] present in test data point [True]
454 Text feature [buffer] present in test data point [True]
465 Text feature [expression] present in test data point [True]
471 Text feature [alleles] present in test data point [True]
490 Text feature [endogenous] present in test data point [True]
491 Text feature [most] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [88]:

```
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 : 1
Predicted Class Probabilities: [[0.4544 0.0753 0.007  0.1482 0.2692 0.0308 0.0124 0.0013 0.0013]]
Actual Class : 5
--------------------------------------------------
54 Text feature [significant] present in test data point [True]
248 Text feature [can] present in test data point [True]
252 Text feature [would] present in test data point [True]
253 Text feature [phenotype] present in test data point [True]
265 Text feature [indicated] present in test data point [True]
267 Text feature [following] present in test data point [True]
271 Text feature [four] present in test data point [True]
277 Text feature [deleterious] present in test data point [True]
281 Text feature [methods] present in test data point [True]
283 Text feature [we] present in test data point [True]
286 Text feature [different] present in test data point [True]
288 Text feature [patient] present in test data point [True]
289 Text feature [did] present in test data point [True]
292 Text feature [classification] present in test data point [True]
296 Text feature [strong] present in test data point [True]
300 Text feature [identify] present in test data point [True]
302 Text feature [positive] present in test data point [True]
304 Text feature [similar] present in test data point [True]
324 Text feature [interactions] present in test data point [True]
325 Text feature [some] present in test data point [True]
```

```
327 Text feature [whether] present in test data point [True]
333 Text feature [cells] present in test data point [True]
334 Text feature [have] present in test data point [True]
337 Text feature [alterations] present in test data point [True]
340 Text feature [domain] present in test data point [True]
349 Text feature [acids] present in test data point [True]
359 Text feature [above] present in test data point [True]
369 Text feature [both] present in test data point [True]
374 Text feature [very] present in test data point [True]
376 Text feature [including] present in test data point [True]
378 Text feature [control] present in test data point [True]
381 Text feature [examined] present in test data point [True]
389 Text feature [showed] present in test data point [True]
396 Text feature [obtained] present in test data point [True]
399 Text feature [risk] present in test data point [True]
407 Text feature [generated] present in test data point [True]
410 Text feature [five] present in test data point [True]
411 Text feature [side] present in test data point [True]
414 Text feature [gene] present in test data point [True]
427 Text feature [changes] present in test data point [True]
430 Text feature [using] present in test data point [True]
438 Text feature [set] present in test data point [True]
446 Text feature [due] present in test data point [True]
448 Text feature [independent] present in test data point [True]
452 Text feature [cases] present in test data point [True]
453 Text feature [yeast] present in test data point [True]
457 Text feature [while] present in test data point [True]
458 Text feature [other] present in test data point [True]
459 Text feature [genetic] present in test data point [True]
469 Text feature [number] present in test data point [True]
473 Text feature [should] present in test data point [True]
479 Text feature [their] present in test data point [True]
485 Text feature [patients] present in test data point [True]
486 Text feature [is] present in test data point [True]
494 Text feature [type] present in test data point [True]
Out of the top  500  features  55 are present in query point
```

4.4. Linear Support Vector Machines

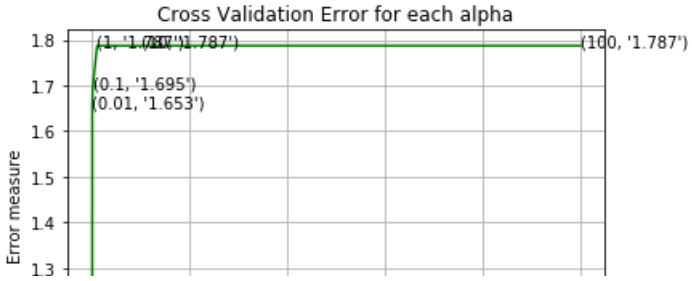
4.4.1. Hyper paramter tuning

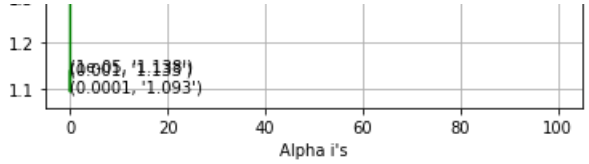
In [89]:

```
# 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(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 : 1.1376142435475454
for C = 0.0001
Log Loss : 1.0926212389570227
for C = 0.001
Log Loss : 1.1326836234952757
for C = 0.01
Log Loss : 1.6533902446182625
for C = 0.1
Log Loss : 1.6946523566805236
for C = 1
Log Loss : 1.7868857579438098
for C = 10
Log Loss : 1.7868861598806443
for C = 100
Log Loss : 1.786886228716208
```





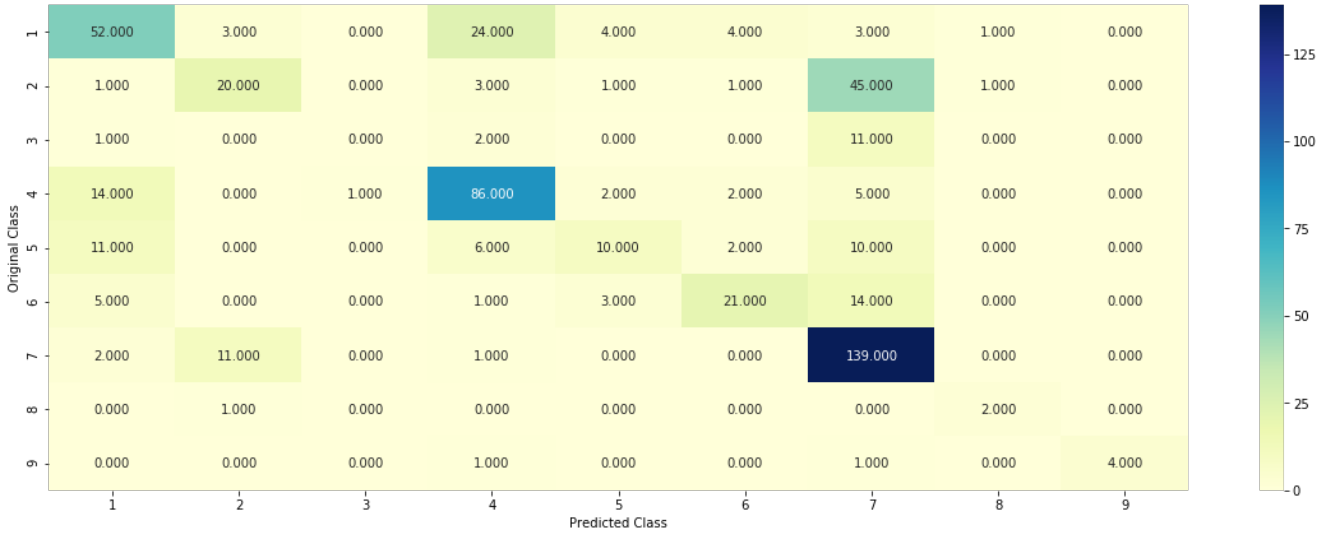

```
For values of best alpha =  0.0001 The train log loss is: 0.4663590247585619
For values of best alpha =  0.0001 The cross validation log loss is: 1.0926212389570227
For values of best alpha =  0.0001 The test log loss is: 1.0584926418941203
```

4.4.2. Testing model with best hyper parameters

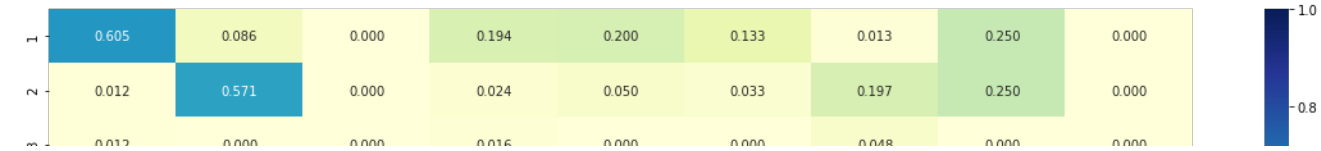
In [90]:

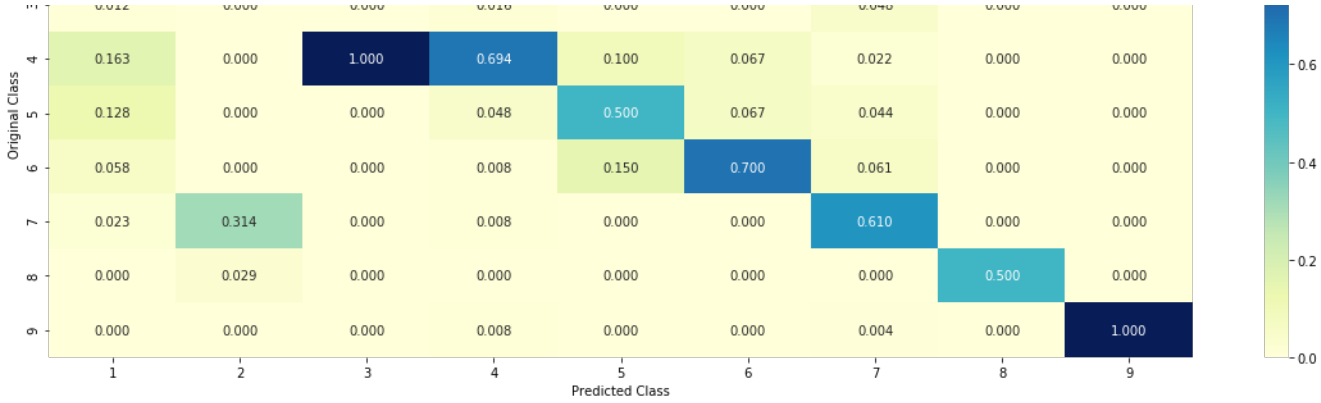
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# 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 : 1.0926212389570227
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

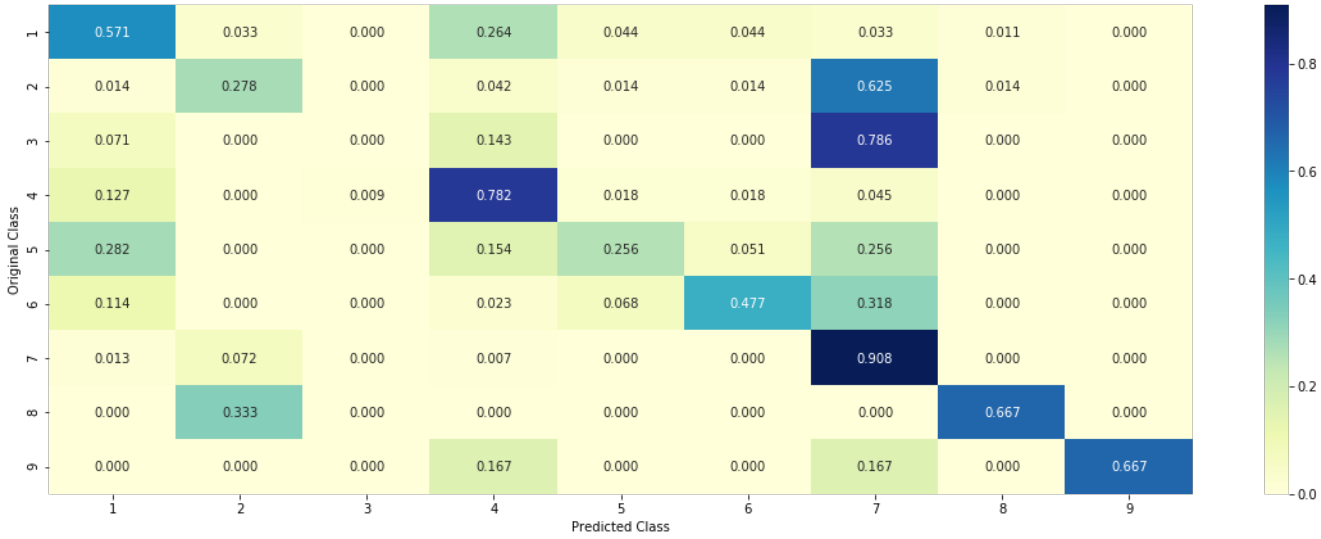



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [91]:

```
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 Probabilities: [[0.0063 0.1422 0.0082 0.0883 0.0282 0.0011 0.721  0.001  0.0038]]
Actual Class : 2
--------------------------------------------------
35 Text feature [alone] present in test data point [True]
36 Text feature [allele] present in test data point [True]
37 Text feature [by] present in test data point [True]
238 Text feature [substitutions] present in test data point [True]
248 Text feature [it] present in test data point [True]
250 Text feature [complete] present in test data point [True]
255 Text feature [lower] present in test data point [True]
```

```
260 Text feature [formation] present in test data point [True]
264 Text feature [type] present in test data point [True]
265 Text feature [because] present in test data point [True]
266 Text feature [buffer] present in test data point [True]
267 Text feature [amino] present in test data point [True]
269 Text feature [endogenous] present in test data point [True]
270 Text feature [other] present in test data point [True]
273 Text feature [suggest] present in test data point [True]
276 Text feature [higher] present in test data point [True]
279 Text feature [signals] present in test data point [True]
280 Text feature [most] present in test data point [True]
284 Text feature [lines] present in test data point [True]
286 Text feature [change] present in test data point [True]
295 Text feature [its] present in test data point [True]
302 Text feature [further] present in test data point [True]
304 Text feature [one] present in test data point [True]
306 Text feature [present] present in test data point [True]
308 Text feature [forms] present in test data point [True]
309 Text feature [in] present in test data point [True]
310 Text feature [effect] present in test data point [True]
311 Text feature [encoding] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [92]:

```
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 : 1
Predicted Class Probabilities: [[6.160e-01 4.630e-02 9.400e-03 1.069e-01 1.669e-01 3.800e-02 1.360
e-02
  6.000e-04 2.300e-03]]
Actual Class : 5
--------------------------------------------------
203 Text feature [can] present in test data point [True]
209 Text feature [significant] present in test data point [True]
211 Text feature [similar] present in test data point [True]
214 Text feature [deleterious] present in test data point [True]
220 Text feature [strong] present in test data point [True]
221 Text feature [indicated] present in test data point [True]
223 Text feature [phenotype] present in test data point [True]
226 Text feature [would] present in test data point [True]
230 Text feature [classification] present in test data point [True]
235 Text feature [did] present in test data point [True]
237 Text feature [whether] present in test data point [True]
238 Text feature [generated] present in test data point [True]
239 Text feature [patient] present in test data point [True]
240 Text feature [interactions] present in test data point [True]
241 Text feature [methods] present in test data point [True]
243 Text feature [above] present in test data point [True]
367 Text feature [following] present in test data point [True]
370 Text feature [independent] present in test data point [True]
375 Text feature [yeast] present in test data point [True]
387 Text feature [examined] present in test data point [True]
389 Text feature [domain] present in test data point [True]
390 Text feature [gene] present in test data point [True]
393 Text feature [obtained] present in test data point [True]
395 Text feature [we] present in test data point [True]
396 Text feature [some] present in test data point [True]
398 Text feature [cases] present in test data point [True]
400 Text feature [have] present in test data point [True]
402 Text feature [high] present in test data point [True]
```

```
409 Text feature [including] present in test data point [True]
417 Text feature [acids] present in test data point [True]
418 Text feature [number] present in test data point [True]
428 Text feature [risk] present in test data point [True]
436 Text feature [showed] present in test data point [True]
442 Text feature [negative] present in test data point [True]
447 Text feature [tested] present in test data point [True]
457 Text feature [an] present in test data point [True]
460 Text feature [positive] present in test data point [True]
461 Text feature [changes] present in test data point [True]
471 Text feature [side] present in test data point [True]
472 Text feature [five] present in test data point [True]
473 Text feature [in] present in test data point [True]
475 Text feature [control] present in test data point [True]
476 Text feature [germline] present in test data point [True]
477 Text feature [four] present in test data point [True]
483 Text feature [cells] present in test data point [True]
498 Text feature [both] present in test data point [True]
499 Text feature [alterations] present in test data point [True]
Out of the top  500  features  47 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [93]:

```
# --------------------------------
# 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)
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 : 1.2825139344967555
for n_estimators = 100 and max depth =  10
Log Loss : 1.3221674058666655
for n_estimators = 200 and max depth =  5
Log Loss : 1.2666768397135757
for n_estimators = 200 and max depth =  10
Log Loss : 1.3041121400853506
for n_estimators = 500 and max depth =  5
Log Loss : 1.2547189231150049
for n_estimators = 500 and max depth =  10
Log Loss : 1.2995438429017439
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2493909600624387
for n_estimators = 1000 and max depth =  10
Log Loss : 1.294523290113384
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2477798291982738
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2959474791842247
For values of best estimator =  2000 The train log loss is: 0.8537243501367902
For values of best estimator =  2000 The cross validation log loss is: 1.2477798291982738
For values of best estimator =  2000 The test log loss is: 1.159298010453977
```

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

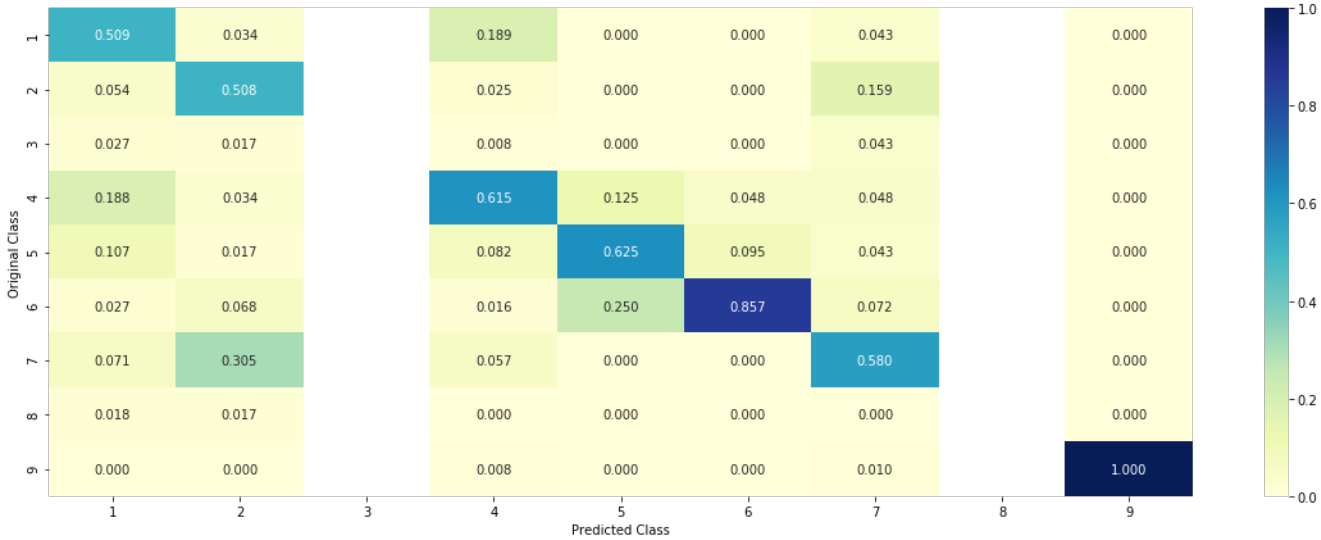
In [94]:

```
# ------------------------------
# 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,
```

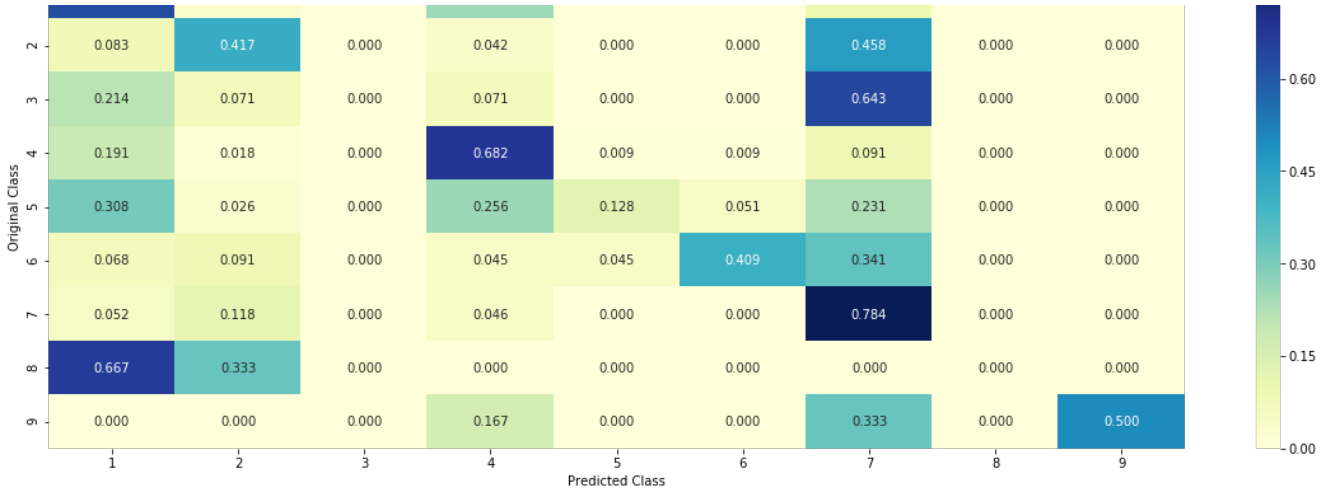
```
verbose=0, warm_start=False,
# class_weight=None)

# 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-forest-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 : 1.2477798291982738
Number of mis-classified points : 0.42105263157894735
-------------------- 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 [95]:

```
# 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: [[0.0994 0.1947 0.026  0.1548 0.0555 0.0485 0.4067 0.011  0.0035]]
Actual Class : 2
--------------------------------------------------
2 Text feature [alleles] present in test data point [True]
3 Text feature [allele] present in test data point [True]
5 Text feature [alone] present in test data point [True]
7 Text feature [group] present in test data point [True]
12 Text feature [its] present in test data point [True]
15 Text feature [difference] present in test data point [True]
20 Text feature [p53] present in test data point [True]
21 Text feature [cancers] present in test data point [True]
26 Text feature [all] present in test data point [True]
29 Text feature [regulatory] present in test data point [True]
30 Text feature [was] present in test data point [True]
31 Text feature [phosphorylation] present in test data point [True]
41 Text feature [like] present in test data point [True]
42 Text feature [forms] present in test data point [True]
44 Text feature [growth] present in test data point [True]
45 Text feature [combination] present in test data point [True]
47 Text feature [bound] present in test data point [True]
51 Text feature [it] present in test data point [True]
52 Text feature [nm] present in test data point [True]
54 Text feature [et] present in test data point [True]
56 Text feature [over] present in test data point [True]
60 Text feature [formation] present in test data point [True]
61 Text feature [one] present in test data point [True]
62 Text feature [developed] present in test data point [True]
67 Text feature [detected] present in test data point [True]
```

```
73 Text feature [present] present in test data point [True]
74 Text feature [compared] present in test data point [True]
76 Text feature [form] present in test data point [True]
77 Text feature [for] present in test data point [True]
82 Text feature [expressed] present in test data point [True]
84 Text feature [or] present in test data point [True]
91 Text feature [are] present in test data point [True]
92 Text feature [receptors] present in test data point [True]
94 Text feature [performed] present in test data point [True]
95 Text feature [then] present in test data point [True]
98 Text feature [indicate] present in test data point [True]
Out of the top  100  features  36 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [96]:

```
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 : 1
Predicted Class Probabilities: [[0.4224 0.0118 0.0174 0.1346 0.234  0.1608 0.0163 0.0019 0.0008]]
Actuall Class : 5
--------------------------------------------------
9 Text feature [previous] present in test data point [True]
10 Text feature [core] present in test data point [True]
12 Text feature [its] present in test data point [True]
15 Text feature [difference] present in test data point [True]
16 Text feature [analyses] present in test data point [True]
19 Text feature [years] present in test data point [True]
25 Text feature [plasmid] present in test data point [True]
26 Text feature [all] present in test data point [True]
30 Text feature [was] present in test data point [True]
41 Text feature [like] present in test data point [True]
42 Text feature [forms] present in test data point [True]
45 Text feature [combination] present in test data point [True]
46 Text feature [changes] present in test data point [True]
48 Text feature [missense] present in test data point [True]
51 Text feature [it] present in test data point [True]
54 Text feature [et] present in test data point [True]
55 Text feature [sensitivity] present in test data point [True]
61 Text feature [one] present in test data point [True]
62 Text feature [developed] present in test data point [True]
66 Text feature [patients] present in test data point [True]
73 Text feature [present] present in test data point [True]
74 Text feature [compared] present in test data point [True]
77 Text feature [for] present in test data point [True]
84 Text feature [or] present in test data point [True]
90 Text feature [showing] present in test data point [True]
91 Text feature [are] present in test data point [True]
94 Text feature [performed] present in test data point [True]
98 Text feature [indicate] present in test data point [True]
Out of the top  100  features  28 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [97]:

```
# -------------------------------
# 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
```

```
# 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)
```

```
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 : 2.1114042736906224
for n_estimators = 10 and max depth =  3
Log Loss : 1.7726456261640275
for n_estimators = 10 and max depth =  5
Log Loss : 1.730148662156447
for n_estimators = 10 and max depth =  10
Log Loss : 2.033461565448742
for n_estimators = 50 and max depth =  2
Log Loss : 1.8280181831751408
for n_estimators = 50 and max depth =  3
Log Loss : 1.533219553847091
for n_estimators = 50 and max depth =  5
Log Loss : 1.4791091278619115
for n_estimators = 50 and max depth =  10
Log Loss : 1.7892734147317428
for n_estimators = 100 and max depth =  2
Log Loss : 1.6475618800376053
for n_estimators = 100 and max depth =  3
Log Loss : 1.5630822179419028
for n_estimators = 100 and max depth =  5
Log Loss : 1.3624596506041504
for n_estimators = 100 and max depth =  10
Log Loss : 1.7788011477010168
for n_estimators = 200 and max depth =  2
Log Loss : 1.71731165068323
for n_estimators = 200 and max depth =  3
Log Loss : 1.5895284079856766
for n_estimators = 200 and max depth =  5
Log Loss : 1.3985195311580796
for n_estimators = 200 and max depth =  10
Log Loss : 1.7918355712660639
for n_estimators = 500 and max depth =  2
Log Loss : 1.7806988349699544
for n_estimators = 500 and max depth =  3
Log Loss : 1.6088514865117287
for n_estimators = 500 and max depth =  5
Log Loss : 1.413967831524512
for n_estimators = 500 and max depth =  10
Log Loss : 1.7422398737452207
for n_estimators = 1000 and max depth =  2
Log Loss : 1.764788059187881
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5991348547760584
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4040081132451465
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7152730593989267
For values of best alpha =  100 The train log loss is: 0.05297970936122253
For values of best alpha =  100 The cross validation log loss is: 1.3624596506041504
For values of best alpha =  100 The test log loss is: 1.2963995214109985
```

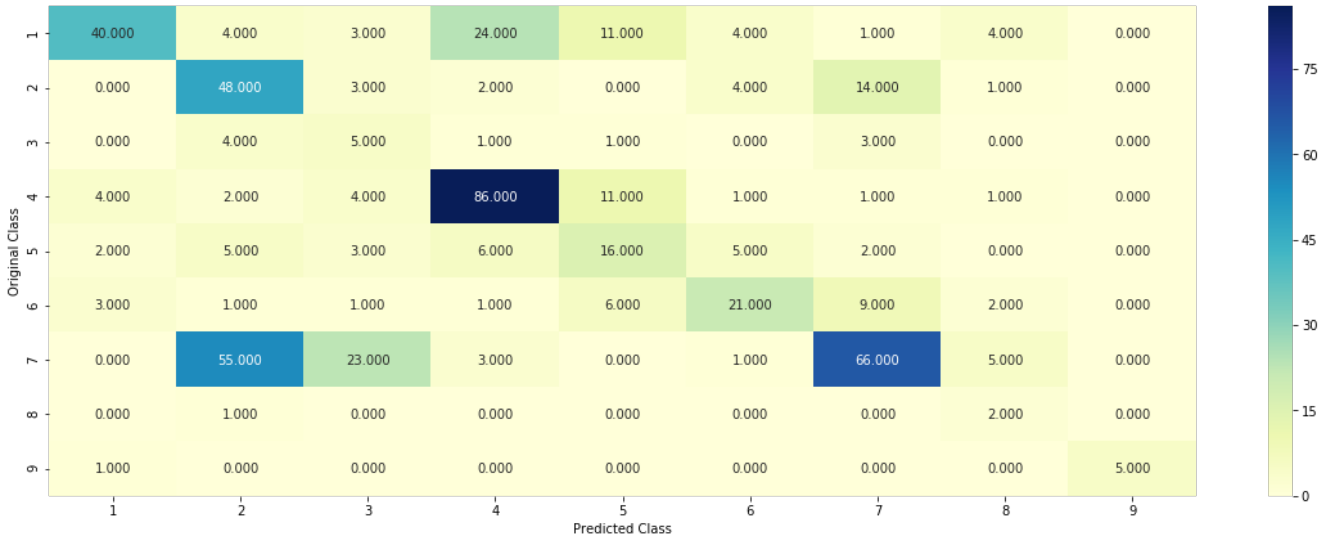
4.5.4. Testing model with best hyper parameters (Response Coding)

In [98]:

```
# ------------------------------
# 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.
```

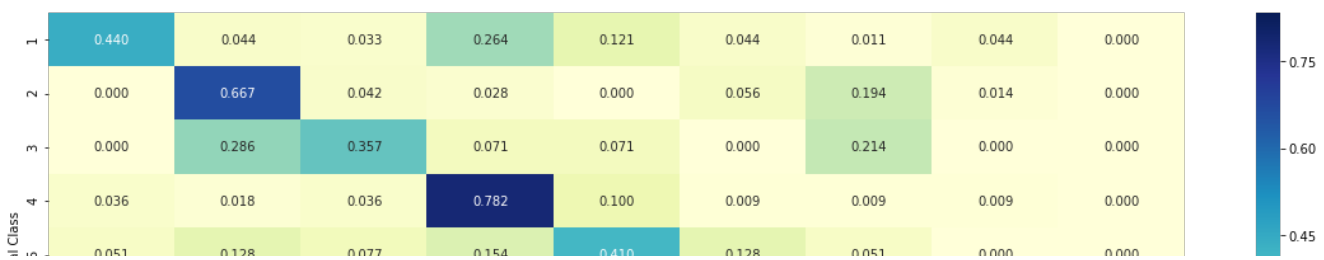
```
# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-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)
```

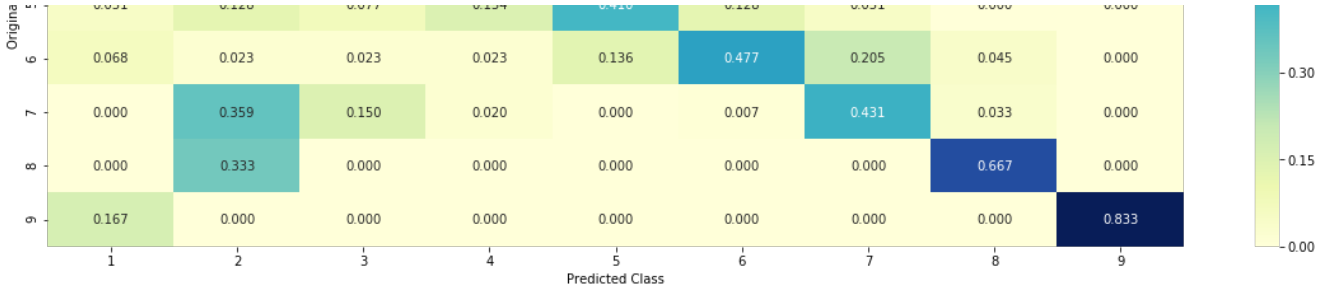
```
Log loss : 1.3624596506041506
Number of mis-classified points : 0.4567669172932331
-------------------- Confusion matrix --------------------
```




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


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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [99]:

```
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 : 7
Predicted Class Probabilities: [[0.0192 0.2116 0.1552 0.0248 0.0429 0.0526 0.4259 0.052  0.0159]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation 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
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene 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
Variation is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [100]:

```
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 : 5
Predicted Class Probabilities: [[0.0639 0.0076 0.1124 0.04   0.5362 0.2236 0.0044 0.0054 0.0065]]
Actual Class : 5
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation 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
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene 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
Variation 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 [101]:

```
# 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.
```

```
# 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/
#--------------------------------


# 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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_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_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/
# ---------------------------------


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_onehotCoding))))
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)
```

```
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
```

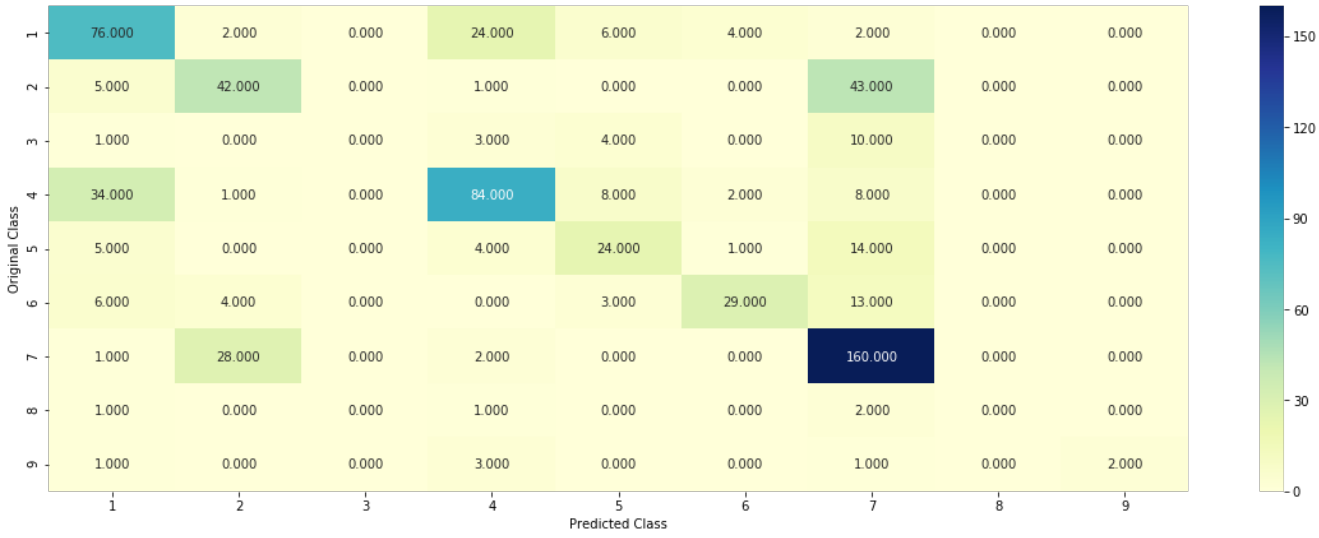
```
Logistic Regression :  Log Loss: 1.12
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.029
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.513
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.252
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.534
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 2.025
```

4.7.2 testing the model with the best hyper parameters

In [102]:

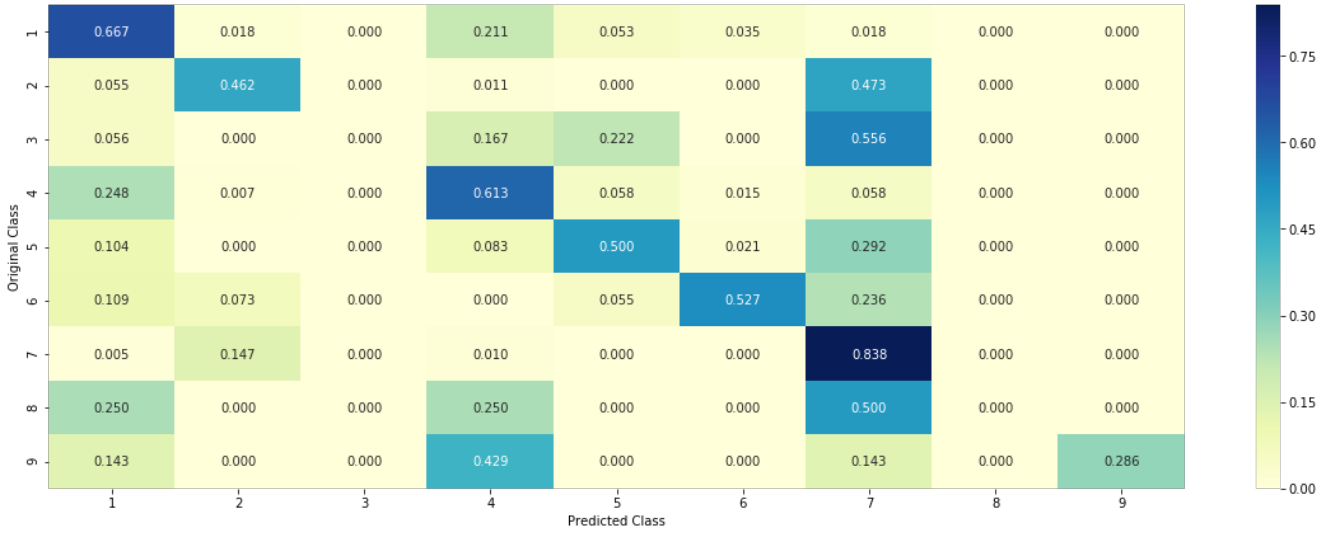
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.5163303446860721
Log loss (CV) on the stacking classifier : 1.2524006047713674
Log loss (test) on the stacking classifier : 1.1654688415700507
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```



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


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

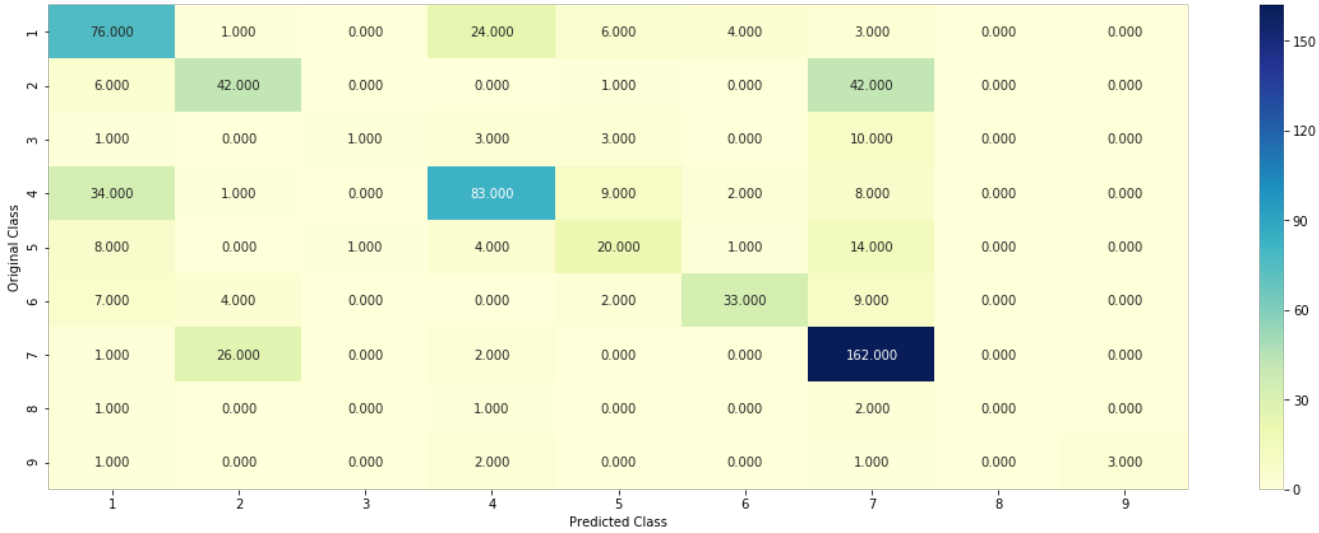


4.7.3 Maximum Voting classifier

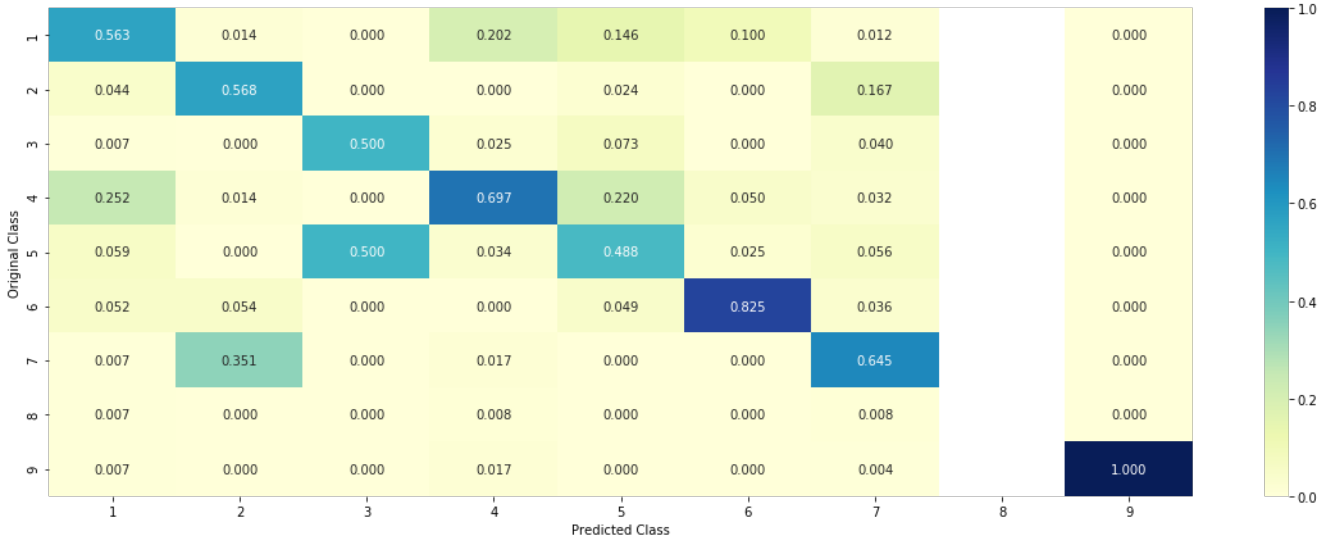
In [103]:

```
#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))
```

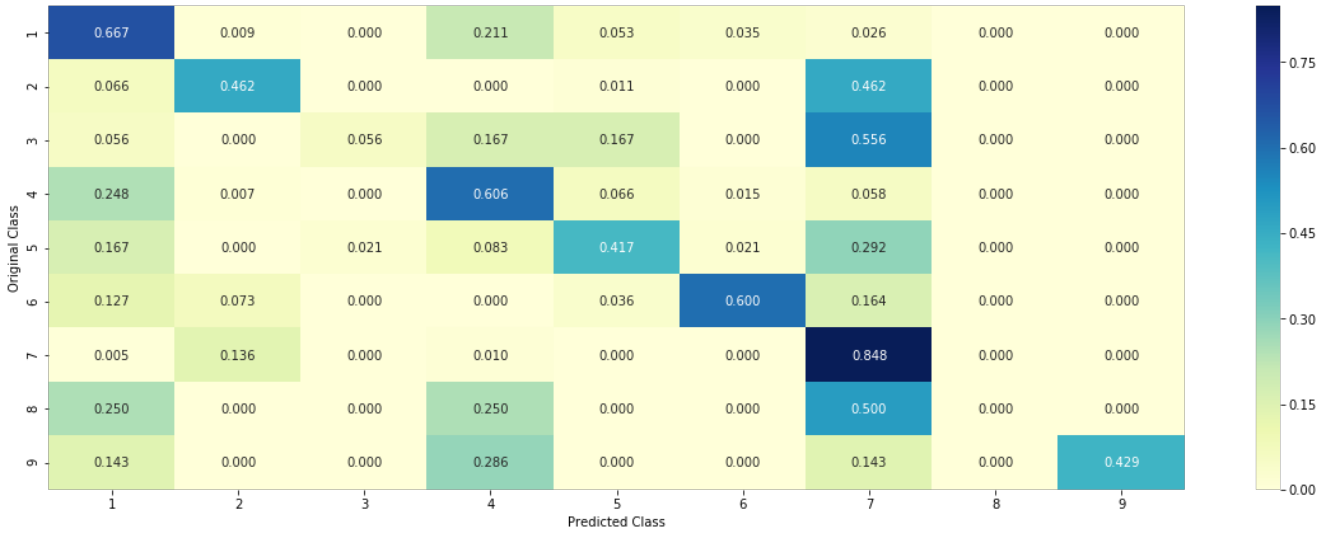
```
Log loss (train) on the VotingClassifier : 0.7927647702025884
Log loss (CV) on the VotingClassifier : 1.2155334323387001
Log loss (test) on the VotingClassifier : 1.1535790239468708
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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



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



Conclusion:

Personalized Cancer Diagnosis Using Tfidfvectorizer with feature engineering 1) WE have taken training variants and training_text

CSV files from kaggle site for personalized cancer diagnosis. #### EDA: 2) In Exploratory Data Analysis, visualize and plot Gene , variation and Class columns for understanding of data. ### Feature Engineering 3) Extract a number of features like Gene_variation feature, number of words, number of characters,average word length. 4) In EDA, for text preprocessing we remove the stopwords, lower the characters. After EDA, we split the data for train,test and cross validation using train_test_split. 5)We plot the distribution of class in each train, test and cross validation, and it seems to equal in all three parts. #### CReate a Random MOdel 6)Next build a random model.In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1. 7) Create a function to plot confusion matrix for Confusion, recall, precision matrix. #### Univariate Analysis 8) For Univariate Analysis, create a function for Gene variation Dictionary, which contains the probability array for each gene/variation and create Gene variation feature, it will contain the feature for each feature value in the data. 9) To caculate the probability of a feature belongs to any particular class, we apply laplace smoothing. ### Univariate Analysis on Gene Feature 10) Observe Gene is a what tye of feature,how many categories are there and how they are distributed. 11) Plot a histogram and cumulative distributive of genes. 12) Featurize the Gene feature using response coding and one hot coding. 13) Observe how good is this gene feature in predicting y_i, here we build Logistic regrewssion for prediction. 14) And compute log loss for train ,cv and test. 14) Observe the Gene feature stable across all the data sets Test, Train, Cross validation. 15) Repete the steps for Variation and Text feture. ### Univariate Analysis on Text Feature 16) Observe, how many unique words are present in train data 17) How are word frequencies distributed? 18) Featurize text field, build a TFidfvectorizer with max 5000 features. 19) Is the text feature useful in predicitng y_i? 20) Is the text feature stable across train, test and CV datasets ### Machine lerning Models 21) Prepare a data for ML models define a function to predict and plot confusion matrix, use calibratedClassifierCV because we want probabilities. 22) Define a function for log loss and important features. 23) Shack the three features ##### Base line models 24) Here we have build naive bayes, knn, LOgistic regression, svm. random forest, stacking and voting classifier. 25) Do the hyperparameter tuning and test the model with best hyperparameter. 26) Note the log loss for train, test and cv.

In [108]:

```
### After Feature Engineering
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","features","Train ","CV","Test","% Missclassified"]

x.add_row(["Naive Bayes", "OneHOtEncoding","0.492","1.244","1.151","41.54%"])
x.add_row(["KNN","Response coding", "0.584","1.155","1.099","38.15%"])
x.add_row(["Logistic(classbalance)", "OneHOtEncoding","0.417","1.089","1.007","37.96%"])
x.add_row(["Logistic(without)","OneHOtEncoding", "0.508","1.144","1.126","38.15%"])
x.add_row(["Linear SVM","OneHOtEncoding", "0.466","1.092","1.058","37.21%"])
x.add_row(["Random Forest", "OneHOtEncoding","0.853","1.247","1.159","42.10%"])
x.add_row(["Random Forest","Response Coding", "0.0529","1.362","1.296","45.67%"])
x.add_row(["Stacking Classifier", "OneHOtEncoding","0.516","1.252","1.165","37.29%"])
x.add_row(["Voting Classifier","OneHOtEncoding", "0.792","1.215","1.153","36.84%"])


print(x)
```

```
+------------------------+-----------------+--------+-------+-------+------------------+
|         Model          |     features    | Train  |   CV  |  Test | % Missclassified |
+------------------------+-----------------+--------+-------+-------+------------------+
|      Naive Bayes       |  OneHOtEncoding | 0.492  | 1.244 | 1.151 |      41.54%      |
|          KNN           | Response coding | 0.584  | 1.155 | 1.099 |      38.15%      |
| Logistic(classbalance) |  OneHOtEncoding | 0.417  | 1.089 | 1.007 |      37.96%      |
|   Logistic(without)    |  OneHOtEncoding | 0.508  | 1.144 | 1.126 |      38.15%      |
|       Linear SVM       |  OneHOtEncoding | 0.466  | 1.092 | 1.058 |      37.21%      |
|     Random Forest      |  OneHOtEncoding | 0.853  | 1.247 | 1.159 |      42.10%      |
|     Random Forest      | Response Coding | 0.0529 | 1.362 | 1.296 |      45.67%      |
|  Stacking Classifier   |  OneHOtEncoding | 0.516  | 1.252 | 1.165 |      37.29%      |
|   Voting Classifier    |  OneHOtEncoding | 0.792  | 1.215 | 1.153 |      36.84%      |
+------------------------+-----------------+--------+-------+-------+------------------+
```

1) After feature engineering For TFIDF vectorizer with top 1000 features we are getting best result with LOgistic regression with one hot coding for class balancing. 2) For TFIDF vectorizer with top 1000 features we are getting worst result with Random Forest with response coding. 3) After feature engineering loss slightly decreases.