

# 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=UmbuW7oK8rk>
3. <https://www.youtube.com/watch?v=qxXRKVompl8>

Mutations are mistakes in cells DNA that ultimately lead to abnormal protein production.\ DNA ----> RNA ----> Protein\ There are nine different classes a genetic mutation can be classified on.\ Class (1-9 the class this genetic mutation has been classified on)\ Point Mutation : When one of our DNA bases is replaced with another, which lead to change in one RNA nucleotide and ultimately a change in one amino acid.\ Frame shift mutation : Instead of changing one base to another , add one to the sequence, this change would lead to an additional base in RNA sequence this change the reading frame of RNA

### 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 contains 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 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 Learning 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 probabilities => 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

### a) Using tf-idf features

## 3. Exploratory Data Analysis

In [0]:

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

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression

/usr/local/lib/python3.6/dist-packages/scikit-learn/utils/deprecation.py:144: FutureWarning: The sklearn.metrics.classification module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.metrics. Anything that cannot be imported from sklearn.metrics is now part of the private API.
    warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/scikit-learn/externals/six.py:31: FutureWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).
    "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/scikit-learn/utils/deprecation.py:144: FutureWarning: The sklearn.nei
```

```
gnbors.base module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the private API.
```

```
warnings.warn(message, FutureWarning)
```

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect\\_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response\\_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:

.....

Mounted at /content/drive

In [0]:

```
data = pd.read_csv('/content/drive/My Drive/cd/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[0]:

	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 [0]:

```
# note the separator in this file
data_text =pd.read_csv("/content/drive/My Drive/cd/training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=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[0]:

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

### 3.1.3. Preprocessing of text

In [0]:

```

import nltk
nltk.download('stopwords')

```

[nltk\_data] Downloading package stopwords to /root/nltk\_data...
[nltk\_data] Unzipping corpora/stopwords.zip.

Out[0]:

True

In [0]:

```

# 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 [0]:

```

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

```

```
there is no text description for id: 1109  
there is no text description for id: 1277  
there is no text description for id: 1407  
there is no text description for id: 1639  
there is no text description for id: 2755  
Time took for preprocessing the text : 25.165107 seconds
```

In [0]:

```
data_text
```

Out[0]:

	ID	TEXT
0	0	cyclin dependent kinases cdks regulate variety...
1	1	abstract background non small cell lung cancer...
2	2	abstract background non small cell lung cancer...
3	3	recent evidence demonstrated acquired uniparen...
4	4	oncogenic mutations monomeric casitas b lineage...
...	...	...
3316	3316	introduction myelodysplastic syndromes mds het...
3317	3317	introduction myelodysplastic syndromes mds het...
3318	3318	runt related transcription factor 1 gene runx1...
3319	3319	runx1 aml1 gene frequent target chromosomal tr...
3320	3320	frequent mutations associated leukemia recurre...

3321 rows × 2 columns

In [0]:

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

Out[0]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineage...

In [0]:

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

Out[0]:

	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	FI T1	Amplification	6	NaN

ID	Gene	Variation	Class	TEXT
2755	2755	BRAF	G596C	7

In [0]:

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

In [0]:

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

Out[0]:

	ID	Gene	Variation	Class	TEXT
	1109	FANCA	S1088F	1	FANCA S1088F

### 3.1.4. Test, Train and Cross Validation Split

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

In [0]:

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

# split the data into test and train by maintaining same distribution of output variable '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 variable '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 [0]:

```
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
```

In [0]:

```
train_df.head(2)
```

Out[0]:

	ID	Gene	Variation	Class	TEXT
1640	1640	FLT3	FLT3_internal_tandem_duplications	7	internal tandem duplications fms like tyrosine...
1045	1045	TSC2	L826P	1	tuberous sclerosis tsc autosomal dominant diso...

#### 3.1.4.2. Distribution of y\_i's in Train, Test and Cross Validation datasets

In [0]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
```

```

# 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_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']

train_class_distribution.plot(kind='bar', color=my_colors)
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.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')

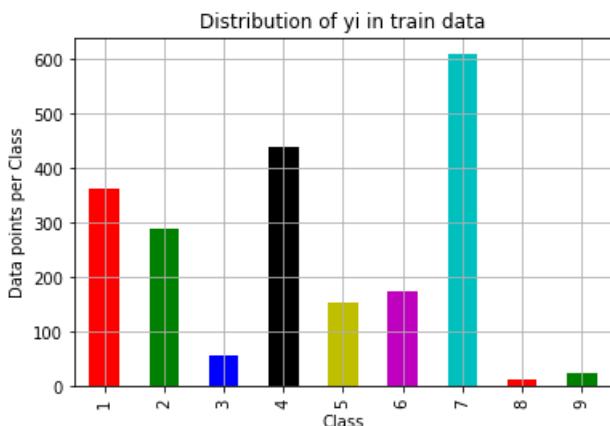
print('*'*80)
#my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# 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.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', cv_class_distribution.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')

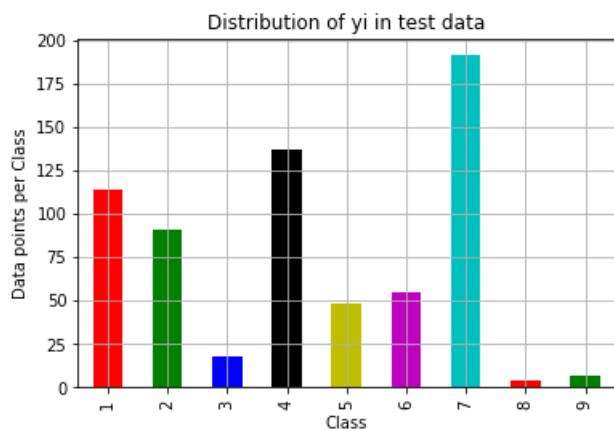
```



Number of data points in class 7 : 609 ( 28.672 %)  
 Number of data points in class 4 : 439 ( 20.669 %)  
 Number of data points in class 1 : 363 ( 17.09 %)  
 Number of data points in class 2 : 289 ( 13.606 %)

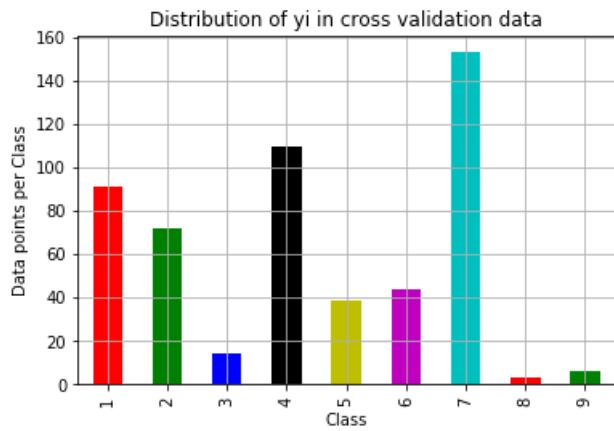
Number of data points in class 6 : 176 ( 8.286 %)  
Number of data points in class 5 : 155 ( 7.298 %)  
Number of data points in class 3 : 57 ( 2.684 %)  
Number of data points in class 9 : 24 ( 1.13 %)  
Number of data points in class 8 : 12 ( 0.565 %)

---



Number of data points in class 7 : 191 ( 28.722 %)  
Number of data points in class 4 : 137 ( 20.602 %)  
Number of data points in class 1 : 114 ( 17.143 %)  
Number of data points in class 2 : 91 ( 13.684 %)  
Number of data points in class 6 : 55 ( 8.271 %)  
Number of data points in class 5 : 48 ( 7.218 %)  
Number of data points in class 3 : 18 ( 2.707 %)  
Number of data points in class 9 : 7 ( 1.053 %)  
Number of data points in class 8 : 4 ( 0.602 %)

---



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

## 3.2 Prediction using a 'Random' Model

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

In [0]:

```
# 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)
```

```

# confusion matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

labels = [1,2,3,4,5,6,7,8,9]
# representing A in heatmap format
print("-"*20, "Confusion matrix", "-"*20)
plt.figure(figsize=(20,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*20, "Precision matrix (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 [0]:

```

rand_probs = np.random.rand(1,9)
print((rand_probs/sum(sum(rand_probs)))[0])

```

```

[0.03987312 0.1790767 0.15703875 0.08993848 0.09830481 0.13110262
 0.16614104 0.05106748 0.087457 ]

```

In [0]:

```

# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])

```

```

print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv, cv_predicted_y, eps=1e-15))

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

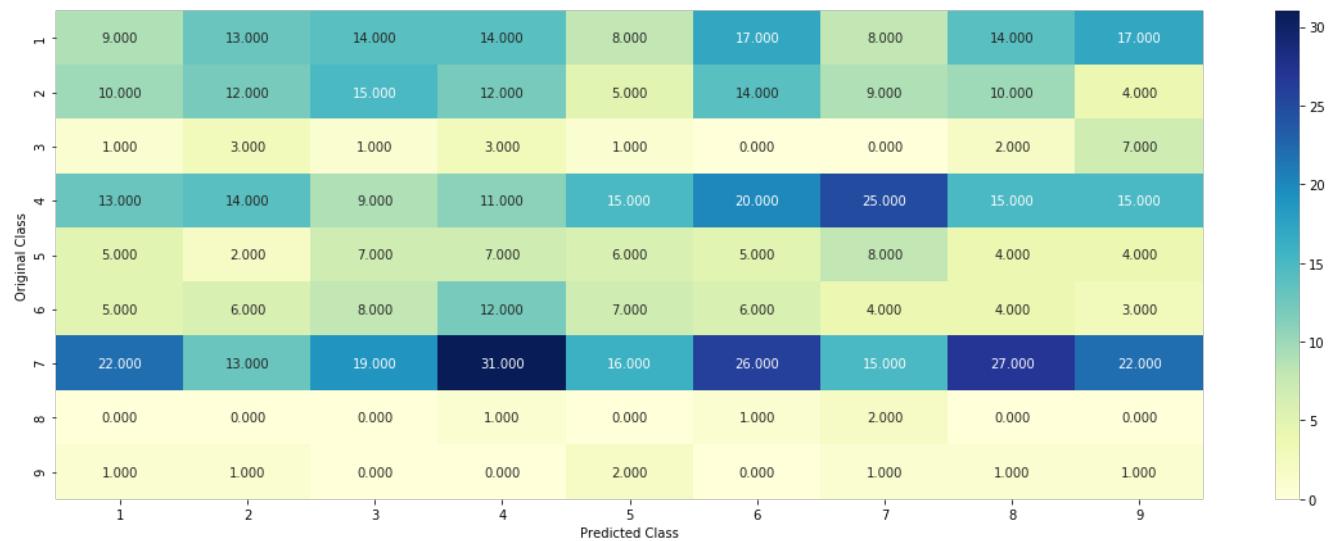
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

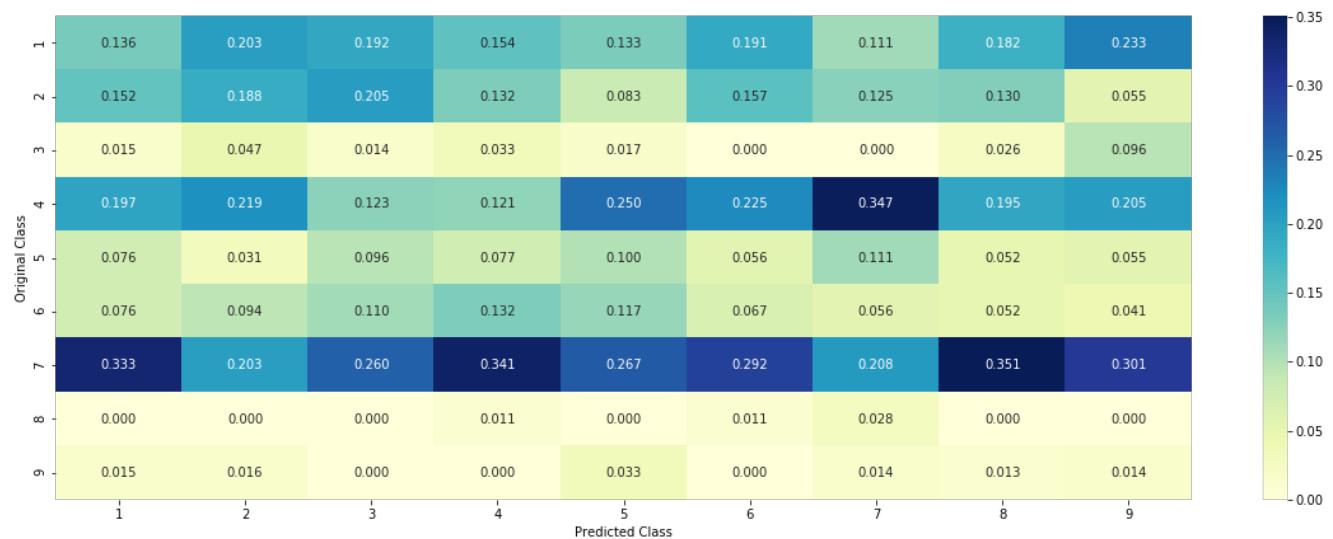
Log loss on Cross Validation Data using Random Model 2.4674949016063987

Log loss on Test Data using Random Model 2.5150188466239616

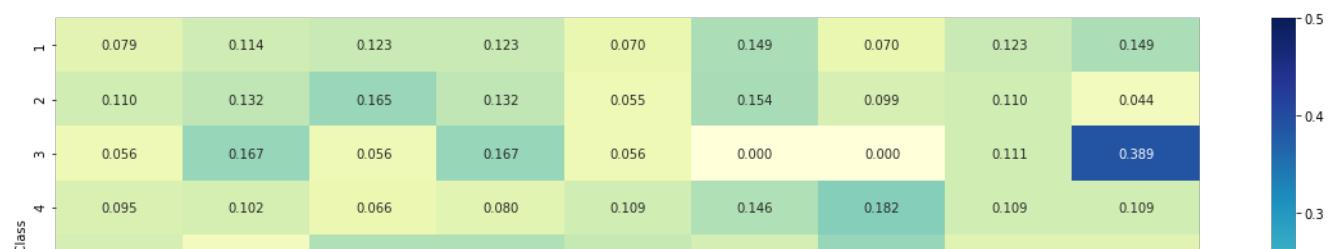
----- Confusion matrix -----

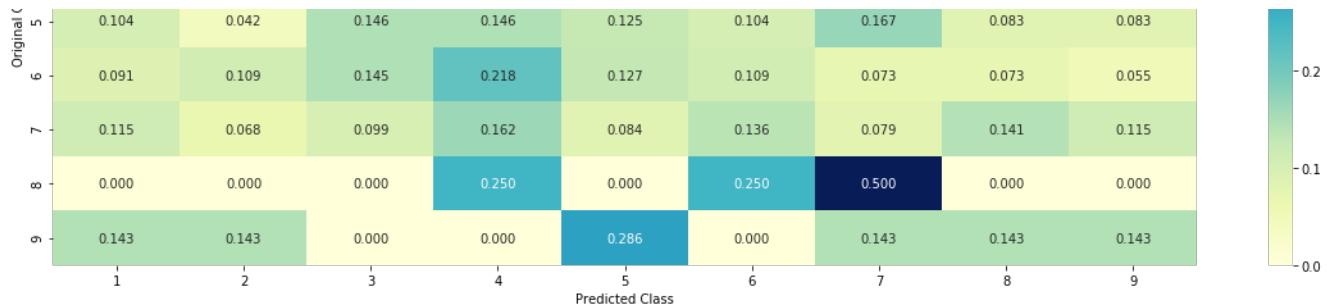


----- Precision matrix (Column Sum=1) -----



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





### 3.3 Univariate Analysis

In [0]:

```
# 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 occurrences of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha / number 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 'gvfea'
# 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] to 'gvfea'
# return 'gvfea'
#
# -----
#
# get_gvfea_dict: Get Gene variation Feature Dict
def get_gvfea_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 occurred in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
        # vec is 9 dimensional vector
        vec = []
        for k in range(1,10):
            vec.append((i==k)/denominator)
        gv_dict[i] = vec

    return gv_dict
```

```

# 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 occurred 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': [0.200757575757575, 0.03787878787878788, 0.0681818181818177, 0.136363636363636363
5, 0.25, 0.193181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653061
5, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    # 'EGFR': [0.0568181818181816, 0.21590909090909091, 0.0625, 0.0681818181818177, 0.06818181
8181818177, 0.0625, 0.34659090909090912, 0.0625, 0.0568181818181816],
    # 'BRCA2': [0.1333333333333333, 0.060606060606060608, 0.060606060606060608, 0.07878787878787878
782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.06060606060
60608],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031446
55, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081
761006289],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284768211920529
5, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556
2913912],
    # 'BRAF': [0.06666666666666666, 0.17999999999999999, 0.07333333333333334, 0.07333333333333333
34, 0.0933333333333338, 0.080000000000000002, 0.29999999999999999, 0.06666666666666666, 0.06666666666
66666666],
    # ...
    # }
    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()

    # gvfea: Gene_variation feature, it will contain the feature for each feature value in the data
    gvfea = []
    # 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 gvfea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gvfea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gvfea.append(gv_dict[row[feature]])
        else:
            gvfea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    #     gvfea.append([-1,-1,-1,-1,-1,-1,-1,-1])
    return gvfea

```

when we calculate the probability of a feature belongs to any particular class, we apply laplace smoothing  
•  $(\text{numerator} + 10 \cdot \text{alpha}) / (\text{denominator} + 90 \cdot \text{alpha})$

### 3.2.1 Univariate Analysis on Gene Feature

**Q1.** Gene, What type of feature it is ?

**Ans.** Gene is a categorical variable

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

In [0]:

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

Number of Unique Genes : 238

Gene	Count
BRCA1	164
TP53	102
EGFR	96
PTEN	92
BRCA2	82
KIT	63
BRAF	56
ERBB2	44
ALK	43
PDGFRA	40

Name: Gene, dtype: int64

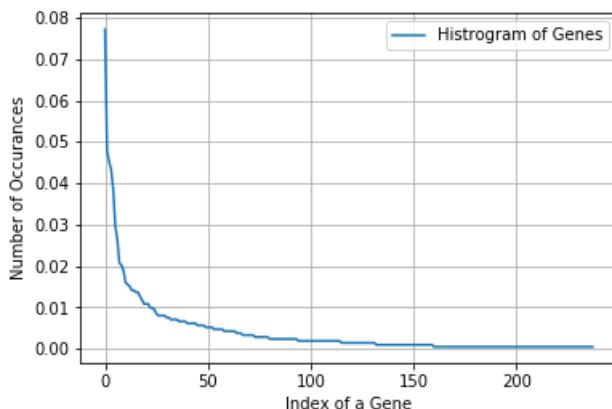
In [0]:

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

Ans: There are 238 different categories of genes in the train data, and they are distributed as follows

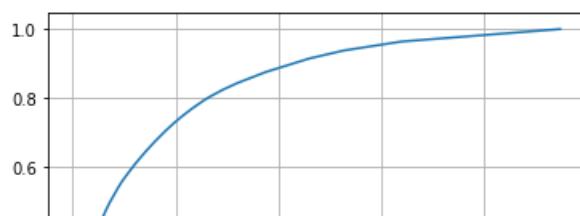
In [0]:

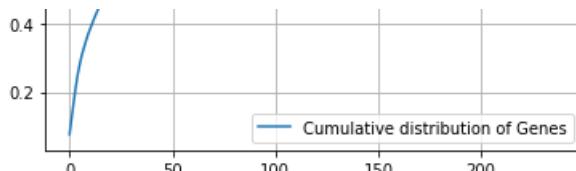
```
s = sum(unique_genes.values);  
h = unique_genes.values/s;  
plt.plot(h, label="Histogram of Genes")  
plt.xlabel('Index of a Gene')  
plt.ylabel('Number of Occurrences')  
plt.legend()  
plt.grid()  
plt.show()
```



In [0]:

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





### Q3. How to featurize this Gene feature ?

**Ans.** there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [0]:

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

In [0]:

```
print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

train\_gene\_feature\_responseCoding is converted feature using response coding method. The shape of gene feature: (2124, 9)

In [0]:

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

In [0]:

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

Out[0]:

```
1640      FLT3
1045      TSC2
1506      ALK
2781      BRCA2
952       PDGFRB
Name: Gene, dtype: object
```

In [0]:

```
gene_vectorizer.get_feature_names()
```

Out[0]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1']
```

'akt1',  
'akt2',  
'akt3',  
'alk',  
'apc',  
'ar',  
'araf',  
'arid1a',  
'arid1b',  
'arid2',  
'arid5b',  
'asxl1',  
'asxl2',  
'atm',  
'atr',  
'atrx',  
'aurka',  
'aurkb',  
'axl',  
'b2m',  
'bap1',  
'bcl10',  
'bcl2',  
'bcl2l11',  
'bcor',  
'braf',  
'brca1',  
'brca2',  
'brd4',  
'brip1',  
'btk',  
'card11',  
'carm1',  
'casp8',  
'cbl',  
'ccnd1',  
'ccnd2',  
'ccnd3',  
'ccne1',  
'cdh1',  
'cdk12',  
'cdk4',  
'cdk6',  
'cdk8',  
'cdkn1a',  
'cdkn1b',  
'cdkn2a',  
'cdkn2b',  
'cdkn2c',  
'cebpA',  
'chek2',  
'cic',  
'crebbp',  
'ctcf',  
'ctla4',  
'ctnnb1',  
'ddr2',  
'dicer1',  
'dnmt3a',  
'dnmt3b',  
'dusp4',  
'egfr',  
'eiflax',  
'elf3',  
'ep300',  
'epas1',  
'erbb2',  
'erbb3',  
'erbb4',  
'ercc2',  
'ercc3',  
'ercc4',  
'erg',  
'esr1',  
'etv1',  
'etv6',  
'ewsrl1',  
'ezh2'.

'  
'fam58a',  
'fanca',  
'fancc',  
'fat1',  
'fbxw7',  
'fgf19',  
'fgf3',  
'fgfr1',  
'fgfr2',  
'fgfr3',  
'fgfr4',  
'flt3',  
'foxa1',  
'foxl2',  
'foxp1',  
'fubp1',  
'gata3',  
'gnas',  
'h3f3a',  
'hla',  
'hnf1a',  
'hras',  
'idh1',  
'idh2',  
'igf1r',  
'ikbke',  
'ikzf1',  
'il7r',  
'inpp4b',  
'jak1',  
'jak2',  
'jun',  
'kdm5a',  
'kdm5c',  
'kdm6a',  
'kdr',  
'keap1',  
'kit',  
'klf4',  
'kmt2a',  
'kmt2b',  
'kmt2c',  
'kmt2d',  
'knstrn',  
'kras',  
'lats1',  
'map2k1',  
'map2k2',  
'map2k4',  
'map3k1',  
'mapk1',  
'mdm4',  
'med12',  
'mef2b',  
'men1',  
'met',  
'mga',  
'mlh1',  
'mpl',  
'msh2',  
'msh6',  
'mtor',  
'myc',  
'mycn',  
'myd88',  
'myod1',  
'nf1',  
'nf2',  
'nfe2l2',  
'nkbia',  
'nkx2',  
'notch1',  
'notch2',  
'npml',  
'nras',  
'nsd1',  
'ntrk1'

'nckx',  
'ntrk2',  
'ntrk3',  
'nup93',  
'pak1',  
'pax8',  
'pbrml1',  
'pdgfra',  
'pdgfrb',  
'pik3ca',  
'pik3cb',  
'pik3r1',  
'pik3r2',  
'pim1',  
'pms2',  
'pole',  
'ppp2rla',  
'ppp6c',  
'prdm1',  
'ptch1',  
'pten',  
'ptpn11',  
'ptprd',  
'ptprt',  
'rab35',  
'rac1',  
'rad21',  
'rad50',  
'rad51b',  
'rad51c',  
'raf1',  
'rara',  
'rasal1',  
'rb1',  
'rbm10',  
'ret',  
'rheb',  
'rhoa',  
'rictor',  
'rit1',  
'rnf43',  
'ros1',  
'rras2',  
'runx1',  
'rxra',  
'rybp',  
'sdhb',  
'setd2',  
'sf3b1',  
'shoc2',  
'smad2',  
'smad3',  
'smad4',  
'smarca4',  
'smarcb1',  
'smo',  
'sos1',  
'sox9',  
'spop',  
'src',  
'srsf2',  
'stag2',  
'stat3',  
'stk11',  
'tcf7l2',  
'tert',  
'tet1',  
'tet2',  
'tgfbri1',  
'tgfbri2',  
'tmpRSS2',  
'tp53',  
'tp53bp1',  
'tsc1',  
'tsc2',  
'u2af1',  
'vegfa',  
'vh1'

```
'vira',
'xpol',
'xrcc2',
'yap1']
```

In [0]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 238)
```

#### Q4. How good is this gene feature in predicting y\_i?

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

In [0]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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:
#-----


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.classes_, 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)

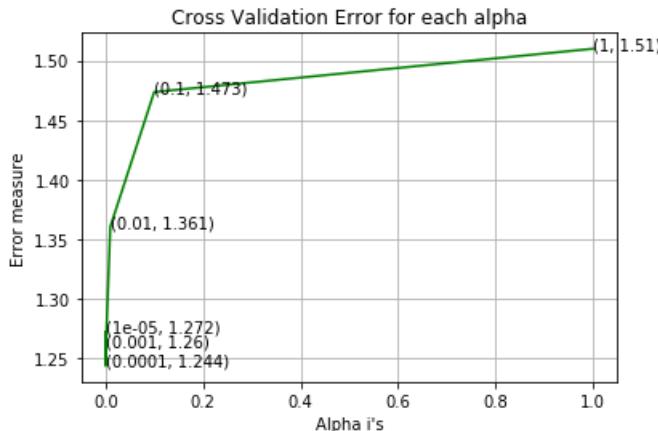
predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
```

```

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_loss(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, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.2720424515711444  
 For values of alpha = 0.0001 The log loss is: 1.2439551948785381  
 For values of alpha = 0.001 The log loss is: 1.260486202566202  
 For values of alpha = 0.01 The log loss is: 1.3605240111042596  
 For values of alpha = 0.1 The log loss is: 1.4734836319770865  
 For values of alpha = 1 The log loss is: 1.5097760181342754



For values of best alpha = 0.0001 The train log loss is: 0.9753696410834307  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.2439551948785381  
 For values of best alpha = 0.0001 The test log loss is: 1.1826890812908906

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

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

In [0]:

```

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

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":" , (test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0]," :" , (cv_coverage/cv_df.shape[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 652 out of 665 : 98.04511278195488
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 [0]:

```

unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])

```

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

```
Number of Unique Variations : 1919
Truncating_Mutations      66
Amplification             49
Deletion                  41
Fusions                   22
Overexpression            6
G12V                      3
Q61R                      3
Q61H                      3
G12S                      2
Promoter_Hypermethylation 2
Name: Variation, dtype: int64
```

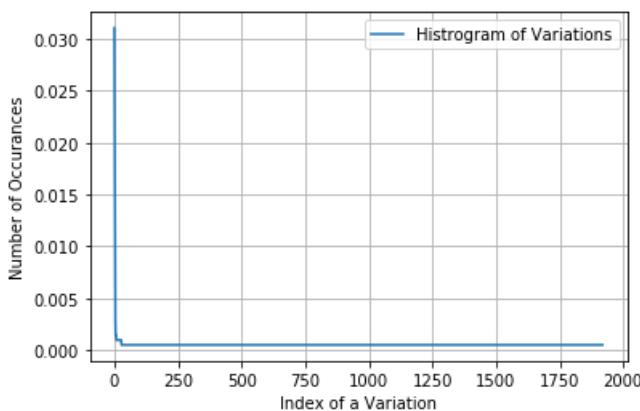
In [0]:

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

Ans: There are 1919 different categories of variations in the train data, and they are distributed as follows

In [0]:

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



In [0]:

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

## 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 [0]:

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

In [0]:

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

train\_variation\_feature\_responseCoding is a converted feature using the response coding method. The shape of Variation feature: (2124, 9)

In [0]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [0]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. T
he 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, 1952)

## Q10. How good is this Variation feature in predicting y\_i?

Let's build a model just like the earlier!

In [0]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, 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.classes_, eps=1e-15))

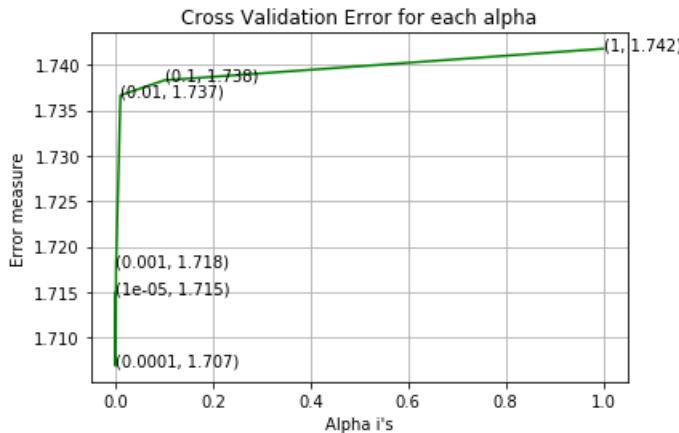
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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_loss(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, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.7149492737634562  
 For values of alpha = 0.0001 The log loss is: 1.7068970151251102  
 For values of alpha = 0.001 The log loss is: 1.7178996323410587  
 For values of alpha = 0.01 The log loss is: 1.7366340041172352  
 For values of alpha = 0.1 The log loss is: 1.7382790320255819  
 For values of alpha = 1 The log loss is: 1.7417486544480132



For values of best alpha = 0.0001 The train log loss is: 0.6944103344825185  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.7068970151251102  
 For values of best alpha = 0.0001 The test log loss is: 1.711979295528625

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

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

In [0]:

```

print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test 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.shape[0])*100)
```

Q12. How many data points are covered by total 1919 genes in test and cross validation data sets?

Ans

1. In test data 65 out of 665 : 9.774436090225564
2. In cross validation data 47 out of 532 : 8.834586466165414

### 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 predicting  $y_i$ ?
5. Is the text feature stable across train, test and CV datasets?

In [0]:

```
# 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 [0]:

```
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 [0]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
train_textfea_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_textfea_counts))

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

In [0]:

```
dict_list = []
# add the words in the list
```

```

# dict_list[i] contains 9 dictionaries 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 build 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 [0]:

```

# response coding of text features
train_text_feature_responseCoding = get_text_responsecoding(train_df)
test_text_feature_responseCoding = get_text_responsecoding(test_df)
cv_text_feature_responseCoding = get_text_responsecoding(cv_df)

```

In [0]:

```

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

```

In [0]:

```

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

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

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

```

In [0]:

```

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

```

In [0]:

```

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

```

In [0]:

```

# Train a Logistic regression+Calibration model using text features which are 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 mo

```

```

# Load more about SGDClassifier, see https://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:
#-----
```

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.classes\_,  
, 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\_loss(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, predict\_y, labels=clf.classes\_, eps=1e-15))

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

**Ans.** Yes, it seems like!

In [0]:

```

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

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

In [0]:

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

## 4. Machine Learning Models

In [0]:

```
#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 will 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 [0]:

```
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 [0]:

```
# 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_imptfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

    feal_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 < feal_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 < feal_len+fea2_len):
            word = var_vec.get_feature_names()[v-(feal_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_no)")
        else:
            word = text_vec.get_feature_names()[v-(feal_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 [0]:

```

# 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)).tocsr()
train_y = np.array(list(train_df['Class']))

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

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

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

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

```

In [0]:

```

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

```

One hot encoding features :

(number of data points \* number of features) in train data = (2124, 56096)  
 (number of data points \* number of features) in test data = (665, 56096)  
 (number of data points \* number of features) in cross validation data = (532, 56096)

In [0]:

```

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

```

Response encoding features :

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

## 4.1. Base Line Model

### 4.1.1. Naive Bayes

#### 4.1.1.1. Hyper parameter tuning

In [0]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=best_alpha)
```

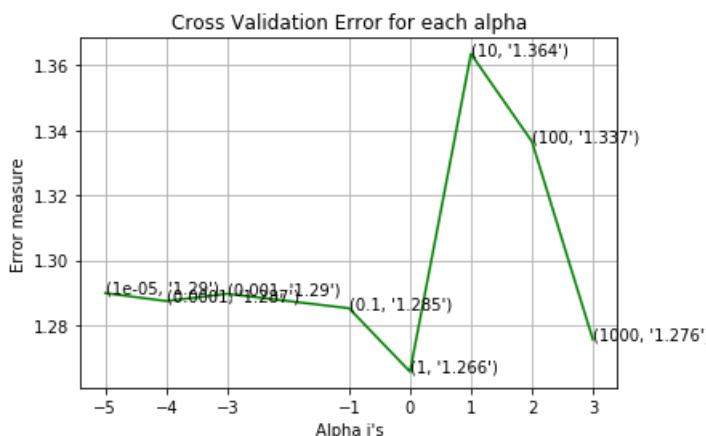
```

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_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-05
Log Loss : 1.2898773682398763
for alpha = 0.0001
Log Loss : 1.2874292546836206
for alpha = 0.001
Log Loss : 1.2896577269471052
for alpha = 0.1
Log Loss : 1.2852084138134348
for alpha = 1
Log Loss : 1.2656762055091821
for alpha = 10
Log Loss : 1.3635766384459984
for alpha = 100
Log Loss : 1.336591192832152
for alpha = 1000
Log Loss : 1.2756113874639805

```



```

For values of best alpha = 1 The train log loss is: 0.8701506219906538
For values of best alpha = 1 The cross validation log loss is: 1.2656762055091821
For values of best alpha = 1 The test log loss is: 1.24546280728209

```

#### 4.1.1.2. Testing the model with best hyper parameters

In [0]:

```

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

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

```

```

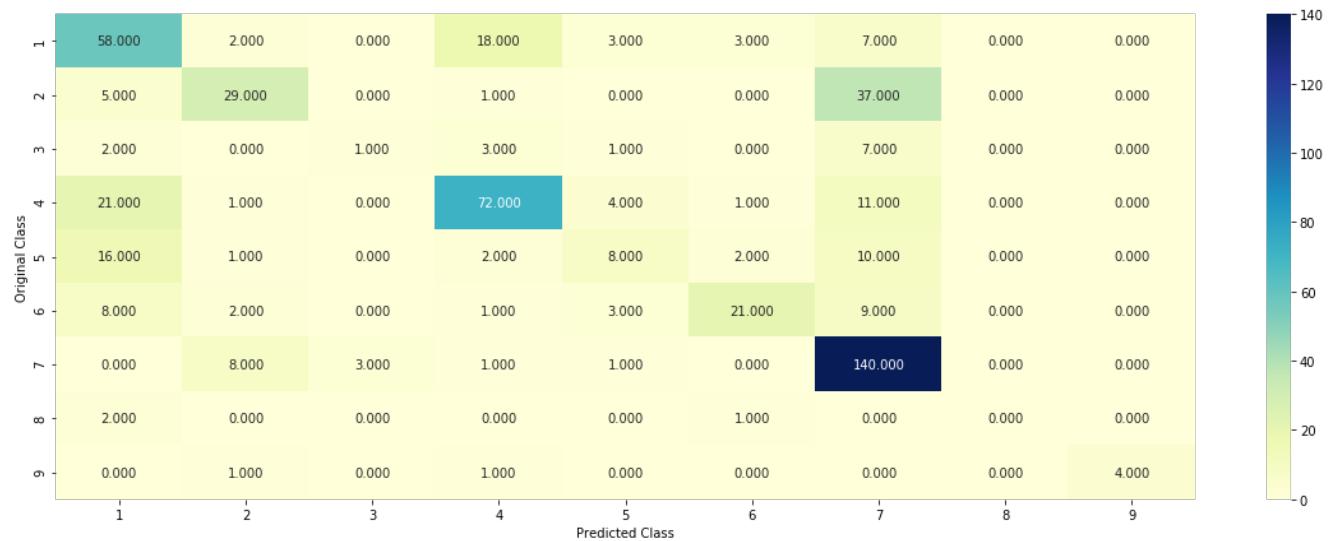
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklear
rn.calibration.CalibratedClassifierCV.html
#
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# 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 probabilités 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()))

```

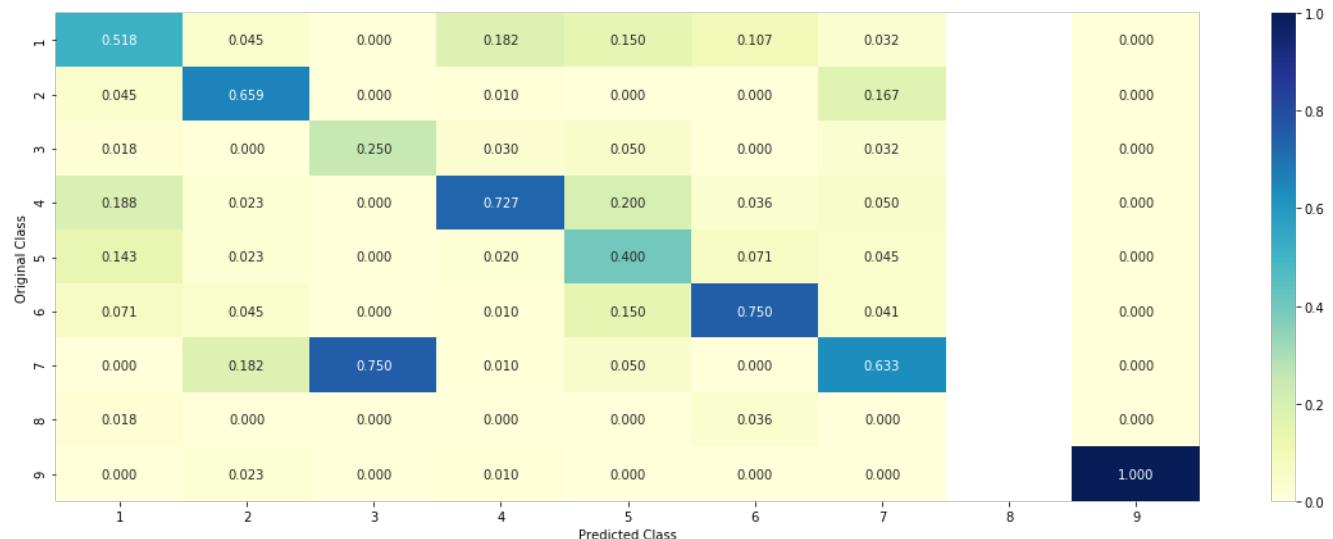
Log Loss : 1.2656762055091821

Number of missclassified point : 0.37406015037593987

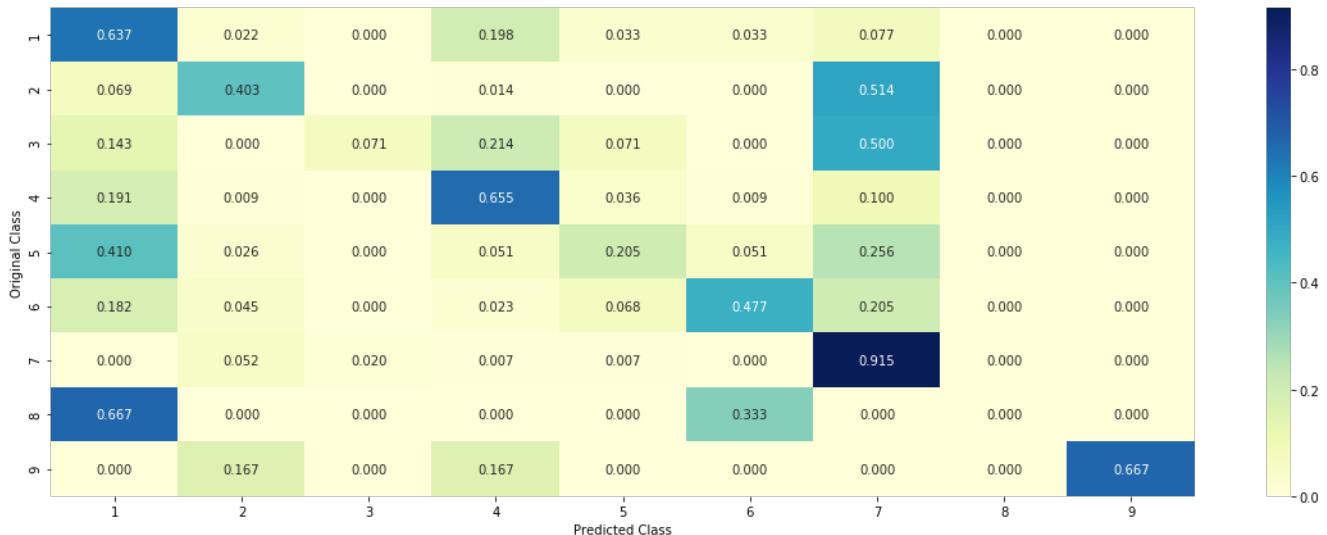
----- Confusion matrix -----



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



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



#### 4.1.1.3. Feature Importance, Correctly classified point

In [0]:

```
test_point_index = 2
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(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_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: [[0.0667 0.0773 0.0212 0.0905 0.0512 0.0374 0.6451 0.006 0.0046]]  
Actual Class : 7

-----  
12 Text feature [cells] present in test data point [True]  
16 Text feature [activated] present in test data point [True]  
17 Text feature [activation] present in test data point [True]  
18 Text feature [cell] present in test data point [True]  
19 Text feature [downstream] present in test data point [True]  
22 Text feature [contrast] present in test data point [True]  
23 Text feature [kinase] present in test data point [True]  
24 Text feature [presence] present in test data point [True]  
25 Text feature [phosphorylation] present in test data point [True]  
26 Text feature [inhibitor] present in test data point [True]  
27 Text feature [shown] present in test data point [True]  
28 Text feature [factor] present in test data point [True]  
29 Text feature [expressing] present in test data point [True]  
30 Text feature [also] present in test data point [True]  
31 Text feature [suggest] present in test data point [True]  
32 Text feature [recently] present in test data point [True]  
33 Text feature [treated] present in test data point [True]  
35 Text feature [10] present in test data point [True]  
36 Text feature [however] present in test data point [True]  
37 Text feature [similar] present in test data point [True]  
38 Text feature [growth] present in test data point [True]  
39 Text feature [found] present in test data point [True]  
41 Text feature [independent] present in test data point [True]  
42 Text feature [tyrosine] present in test data point [True]  
43 Text feature [previously] present in test data point [True]  
44 Text feature [mutant] present in test data point [True]  
45 Text feature [addition] present in test data point [True]  
46 Text feature [signaling] present in test data point [True]  
47 Text feature [constitutive] present in test data point [True]  
48 Text feature [treatment] present in test data point [True]  
49 Text feature [potential] present in test data point [True]

```

50 Text feature [3b] present in test data point [True]
51 Text feature [inhibition] present in test data point [True]
52 Text feature [compared] present in test data point [True]
53 Text feature [1a] present in test data point [True]
54 Text feature [well] present in test data point [True]
55 Text feature [higher] present in test data point [True]
56 Text feature [showed] present in test data point [True]
57 Text feature [increased] present in test data point [True]
58 Text feature [figure] present in test data point [True]
59 Text feature [constitutively] present in test data point [True]
60 Text feature [interestingly] present in test data point [True]
61 Text feature [fig] present in test data point [True]
62 Text feature [mutations] present in test data point [True]
63 Text feature [described] present in test data point [True]
64 Text feature [3a] present in test data point [True]
65 Text feature [followed] present in test data point [True]
66 Text feature [consistent] present in test data point [True]
69 Text feature [furthermore] present in test data point [True]
70 Text feature [may] present in test data point [True]
71 Text feature [pathways] present in test data point [True]
72 Text feature [including] present in test data point [True]
73 Text feature [total] present in test data point [True]
74 Text feature [sensitive] present in test data point [True]
75 Text feature [1b] present in test data point [True]
77 Text feature [using] present in test data point [True]
78 Text feature [demonstrated] present in test data point [True]
79 Text feature [serum] present in test data point [True]
80 Text feature [reported] present in test data point [True]
81 Text feature [obtained] present in test data point [True]
82 Text feature [inhibitors] present in test data point [True]
84 Text feature [confirmed] present in test data point [True]
85 Text feature [various] present in test data point [True]
86 Text feature [observed] present in test data point [True]
87 Text feature [small] present in test data point [True]
88 Text feature [antibodies] present in test data point [True]
89 Text feature [respectively] present in test data point [True]
90 Text feature [expression] present in test data point [True]
91 Text feature [expressed] present in test data point [True]
92 Text feature [new] present in test data point [True]
94 Text feature [activating] present in test data point [True]
95 Text feature [suggesting] present in test data point [True]
97 Text feature [12] present in test data point [True]
99 Text feature [absence] present in test data point [True]
Out of the top 100 features 74 are present in query point

```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

In [0]:

```

test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_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: [[0.6081 0.0849 0.0231 0.0995 0.0562 0.041 0.0755 0.0066 0.0049]]
Actual Class : 1
-----
10 Text feature [type] present in test data point [True]
11 Text feature [protein] present in test data point [True]
12 Text feature [wild] present in test data point [True]
13 Text feature [one] present in test data point [True]
15 Text feature [dna] present in test data point [True]
16 Text feature [results] present in test data point [True]
17 Text feature [two] present in test data point [True]
18 Text feature [therefore] present in test data point [True]
20 Text feature [affect] present in test data point [True]
21 Text feature [functional] present in test data point [True]

```

```

21 Text feature [function] present in test data point [True]
22 Text feature [binding] present in test data point [True]
23 Text feature [involved] present in test data point [True]
24 Text feature [functions] present in test data point [True]
25 Text feature [also] present in test data point [True]
26 Text feature [either] present in test data point [True]
27 Text feature [table] present in test data point [True]
28 Text feature [region] present in test data point [True]
30 Text feature [possible] present in test data point [True]
31 Text feature [role] present in test data point [True]
32 Text feature [using] present in test data point [True]
33 Text feature [loss] present in test data point [True]
34 Text feature [transcriptional] present in test data point [True]
35 Text feature [human] present in test data point [True]
36 Text feature [determined] present in test data point [True]
37 Text feature [present] present in test data point [True]
38 Text feature [three] present in test data point [True]
39 Text feature [effect] present in test data point [True]
40 Text feature [however] present in test data point [True]
42 Text feature [reporter] present in test data point [True]
43 Text feature [ability] present in test data point [True]
44 Text feature [indicate] present in test data point [True]
45 Text feature [amino] present in test data point [True]
46 Text feature [control] present in test data point [True]
48 Text feature [four] present in test data point [True]
49 Text feature [reduced] present in test data point [True]
51 Text feature [may] present in test data point [True]
52 Text feature [indicated] present in test data point [True]
54 Text feature [remaining] present in test data point [True]
58 Text feature [many] present in test data point [True]
59 Text feature [gene] present in test data point [True]
60 Text feature [well] present in test data point [True]
61 Text feature [several] present in test data point [True]
65 Text feature [within] present in test data point [True]
66 Text feature [discussion] present in test data point [True]
68 Text feature [even] present in test data point [True]
69 Text feature [analysis] present in test data point [True]
70 Text feature [transcription] present in test data point [True]
71 Text feature [structure] present in test data point [True]
72 Text feature [including] present in test data point [True]
73 Text feature [addition] present in test data point [True]
74 Text feature [15] present in test data point [True]
75 Text feature [proteins] present in test data point [True]
77 Text feature [likely] present in test data point [True]
78 Text feature [previous] present in test data point [True]
79 Text feature [multiple] present in test data point [True]
82 Text feature [corresponding] present in test data point [True]
83 Text feature [contains] present in test data point [True]
84 Text feature [respectively] present in test data point [True]
86 Text feature [described] present in test data point [True]
88 Text feature [25] present in test data point [True]
89 Text feature [10] present in test data point [True]
92 Text feature [based] present in test data point [True]
93 Text feature [used] present in test data point [True]
94 Text feature [previously] present in test data point [True]
95 Text feature [suggest] present in test data point [True]
97 Text feature [observed] present in test data point [True]
99 Text feature [example] present in test data point [True]
Out of the top 100 features 67 are present in query point

```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

In [0]:

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

```

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


# 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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

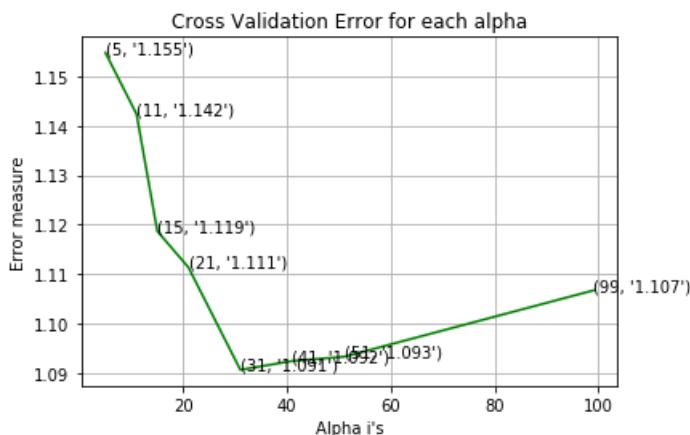
for alpha = 5
Log Loss : 1.1547894528720166
for alpha = 11
Log Loss : 1.1423469309899048
for alpha = 15
Log Loss : 1.1186530396464378
for alpha = 21
Log Loss : 1.1113106903686607
for alpha = 31
Log Loss : 1.0905699324025735
for alpha = 41

```

```

    top alpha = 51
Log Loss : 1.0923645155403867
for alpha = 51
Log Loss : 1.0932272494758484
for alpha = 99
Log Loss : 1.1066480911532348

```



```

For values of best alpha = 31 The train log loss is: 0.8211211074814525
For values of best alpha = 31 The cross validation log loss is: 1.0905699324025735
For values of best alpha = 31 The test log loss is: 1.049252036716099

```

## 4.2.2. Testing the model with best hyper parameters

In [0]:

```

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

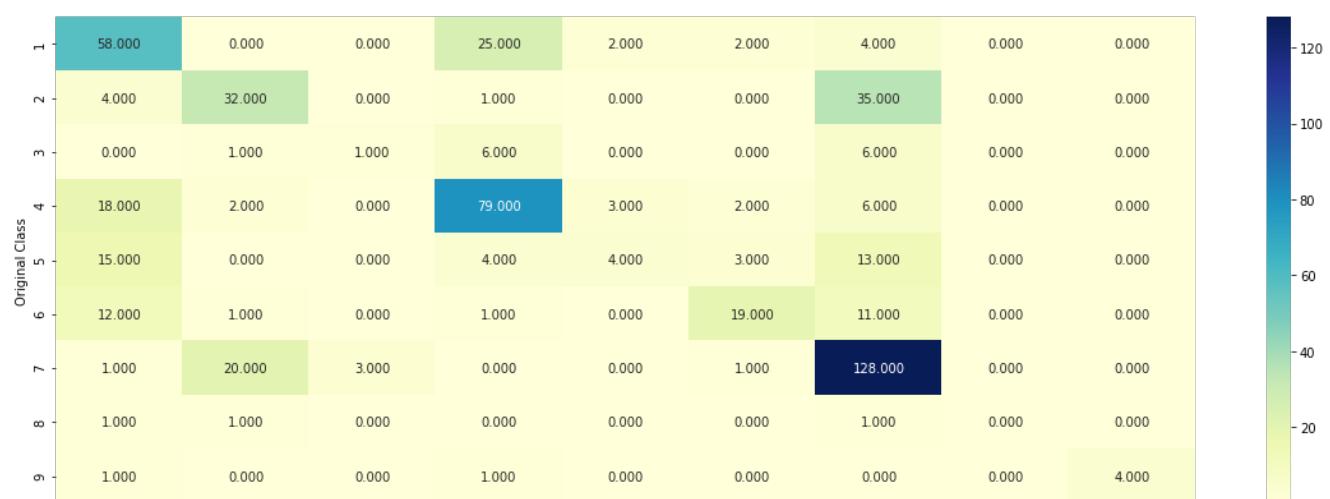
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X) : Predict the class labels for the provided data
# predict_proba(X) : Return probability estimates for the test data X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)

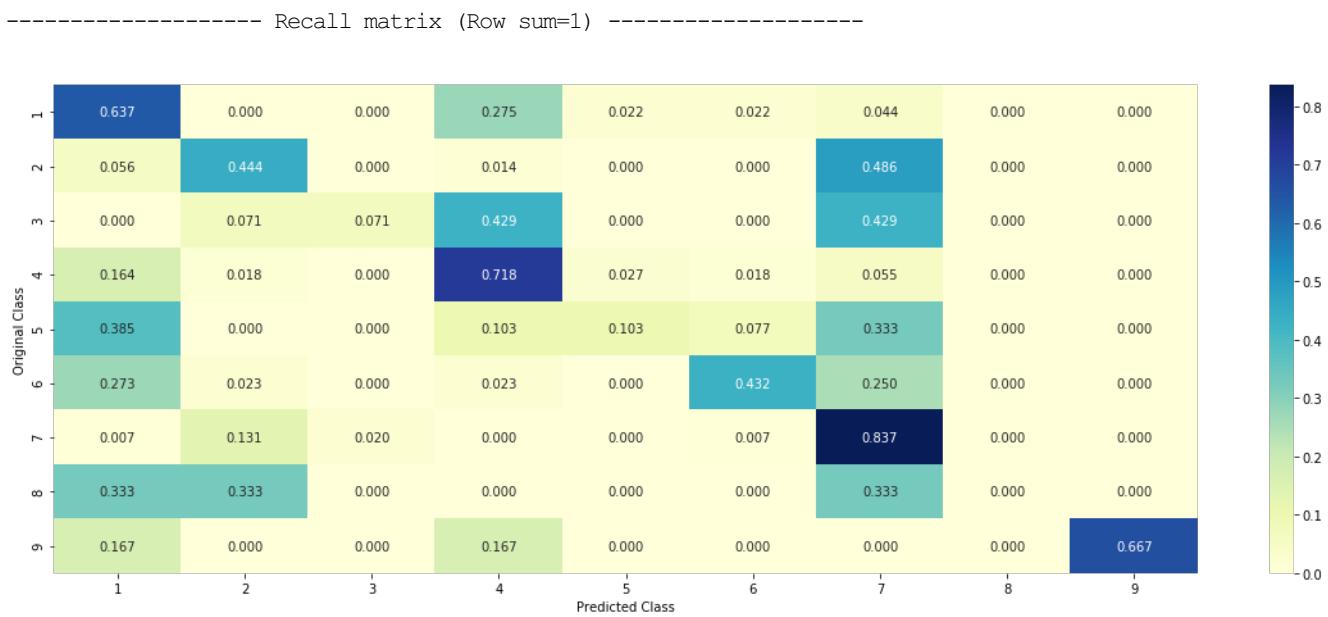
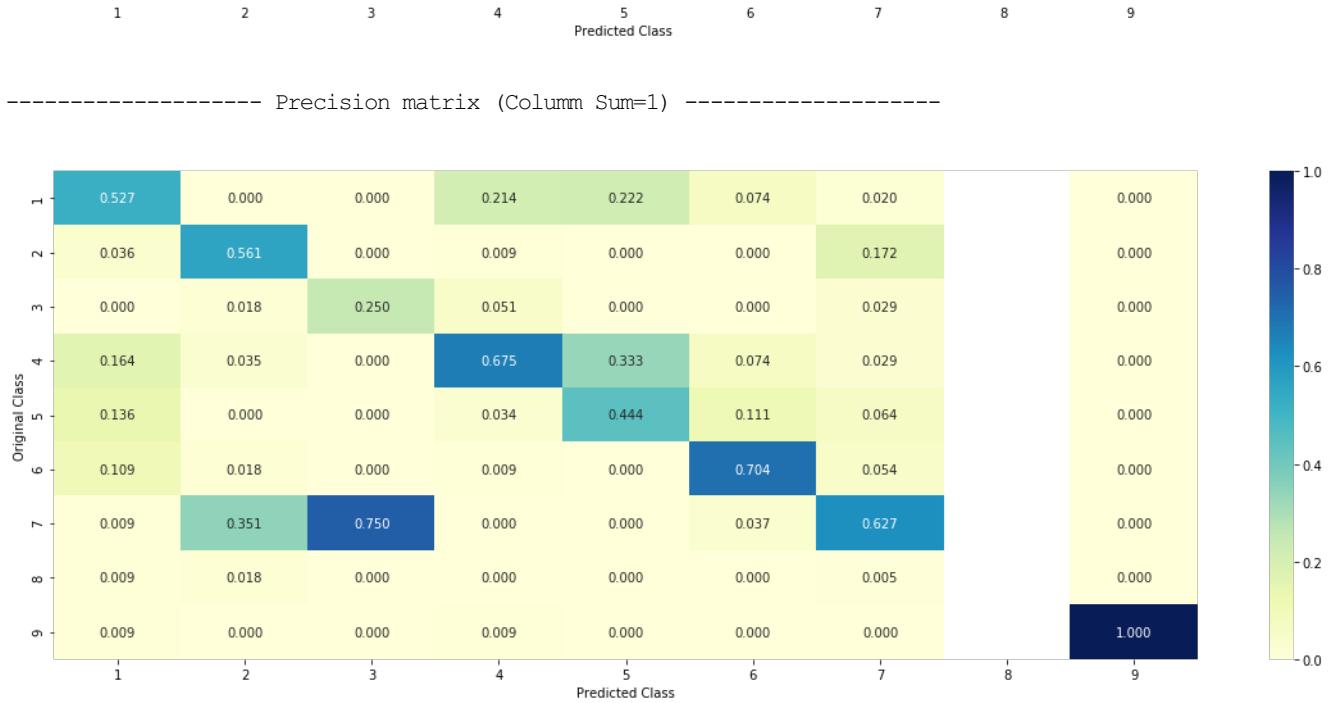
```

```

Log loss : 1.0905699324025735
Number of mis-classified points : 0.3890977443609023
----- Confusion matrix -----

```





### **4.2.3. Sample Query point -1**

In [0]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[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 ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Frequency of nearest points : ",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 7

Actual Class : 2

The 31 nearest neighbours of the test points belongs to classes [7 7 2 2 7 2 2 2 7 2 7 7 2 2 7 7 7 2 2 2 7 7 7 2 2 2 7 7 7 2]

Frequency of nearest points : Counter({2: 16, 7: 15})

#### 4.2.4. Sample Query Point-2

In [0]:

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

test_point_index = 100

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

```
Predicted Class : 1
Actual Class : 1
the k value for knn is 31 and the nearest neighbours of the test points belongs to classes [1 1 1 6 6 6 4 1 1 1 1 6 4 5 6 6 1 5 1 1 1 5 4 4 5 1 1 1 1 1]
Frequency of nearest points : Counter({1: 17, 6: 6, 4: 4, 5: 4})
```

### 4.3. Logistic Regression

#### 4.3.1. With Class balancing

##### 4.3.1.1. Hyper parameter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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/
#-----


# 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.xscale('log')
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', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

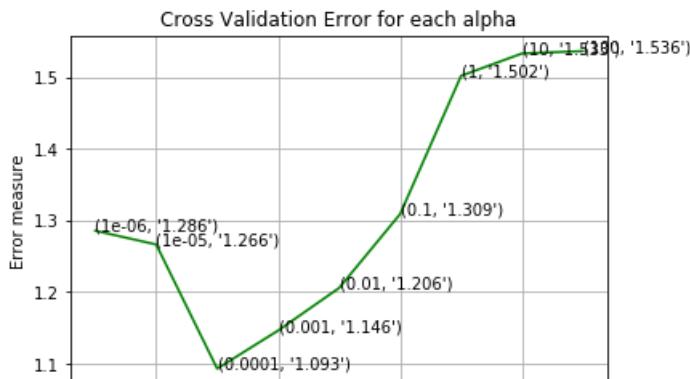
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "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 : 1.2856117850832385
for alpha = 1e-05
Log Loss : 1.266397591641595
for alpha = 0.0001
Log Loss : 1.0932854711611213
for alpha = 0.001
Log Loss : 1.146416289239877
for alpha = 0.01
Log Loss : 1.2057789394729048
for alpha = 0.1
Log Loss : 1.3088159296422197
for alpha = 1
Log Loss : 1.501628472441631
for alpha = 10
Log Loss : 1.5327864487795353
for alpha = 100
Log Loss : 1.5360533611327445

```



$10^{-5}$        $10^{-3}$        $10^{-1}$        $10^1$   
Alpha's

For values of best alpha = 0.0001 The train log loss is: 0.4622803112956112  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.0932854711611213  
 For values of best alpha = 0.0001 The test log loss is: 1.077235378857801

#### 4.3.1.2. Testing the model with best hyper parameters

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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)

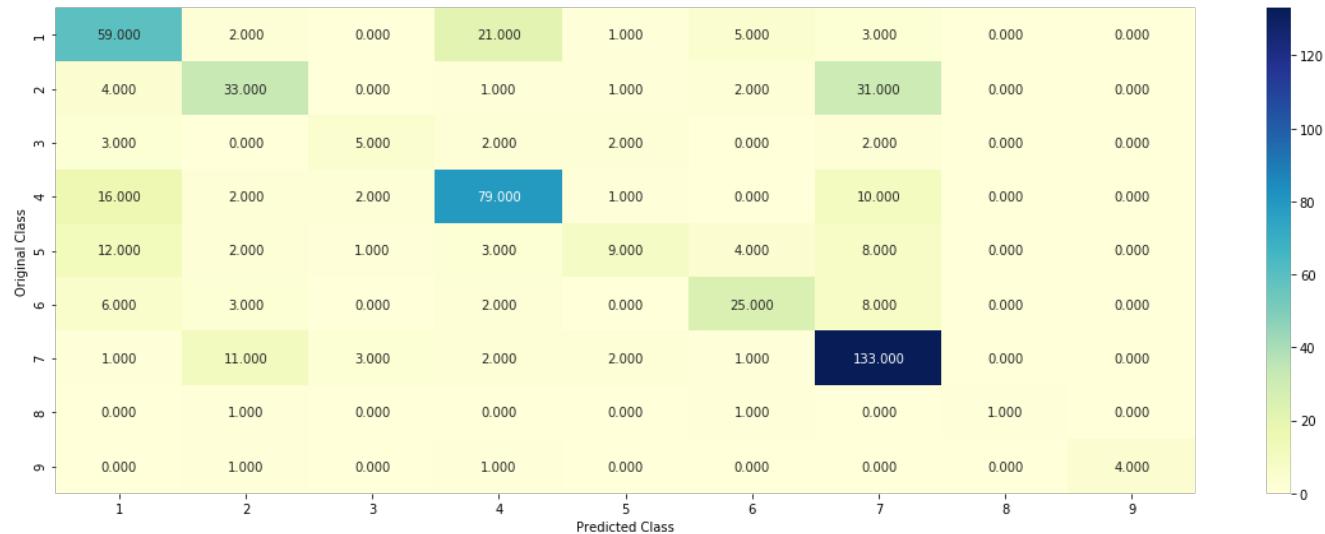
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

#-----
# 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)
```

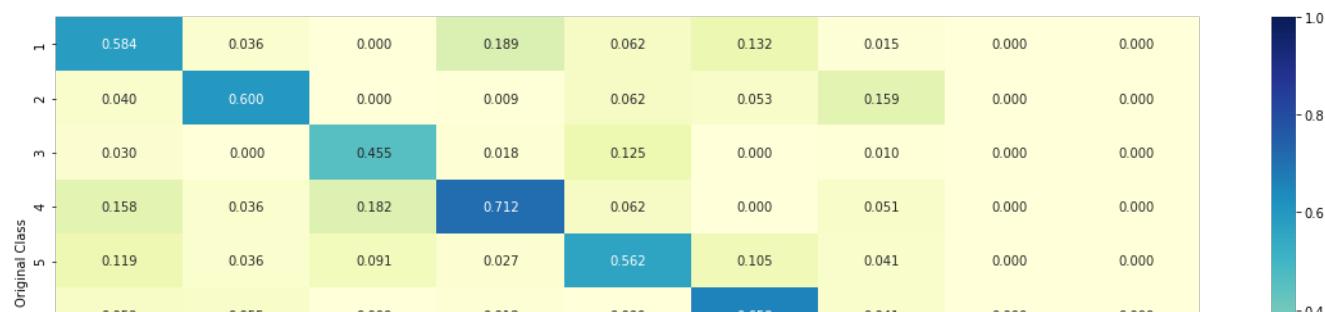
Log loss : 1.0932854711611213

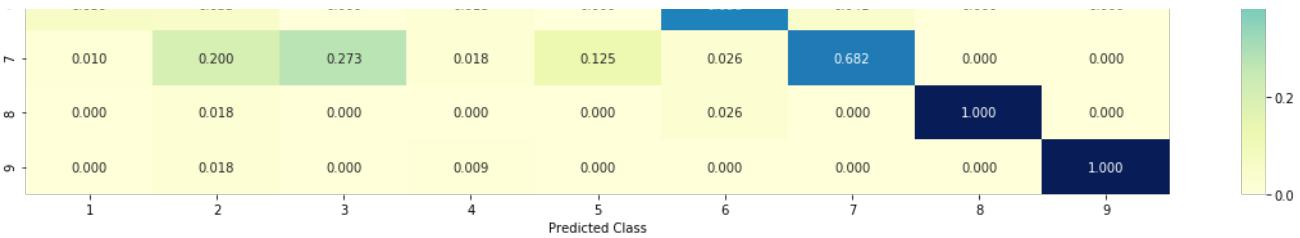
Number of mis-classified points : 0.3458646616541353

----- Confusion matrix -----

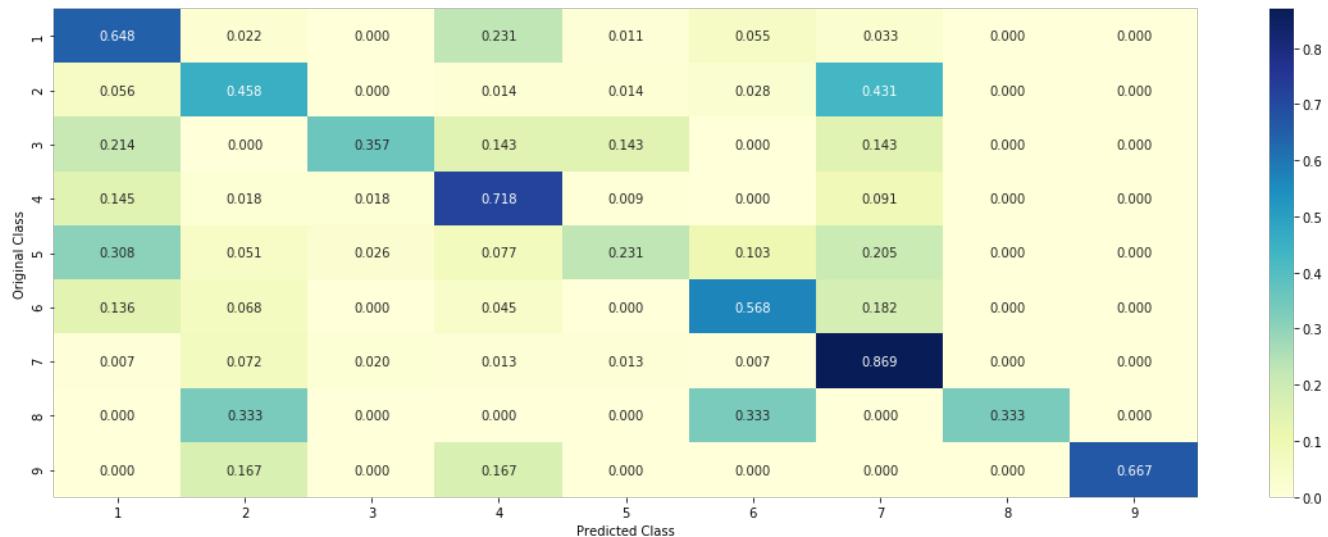


----- Precision matrix (Column Sum=1) -----





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



#### 4.3.1.3. Feature Importance

In [0]:

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

##### 4.3.1.3.1. Correctly Classified point

In [0]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', 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(np.abs(-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: [[0.1036 0.2437 0.0262 0.1391 0.0647 0.075 0.3316 0.005 0.0111]]  
 Actual Class : 2  
 -----  
 Out of the top 500 features 0 are present in query point

#### 4.3.1.3.2. Incorrectly Classified point

In [0]:

```

test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(np.abs(-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: [[0.752 0.0179 0.0033 0.0822 0.0327 0.0947 0.0074 0.0053 0.0045]]  
 Actual Class : 1  
 -----  
 9 Text feature [given] present in test data point [True]  
 18 Text feature [alleles] present in test data point [True]  
 67 Text feature [functional] present in test data point [True]  
 287 Text feature [example] present in test data point [True]  
 411 Text feature [nh2] present in test data point [True]  
 430 Text feature [indicates] present in test data point [True]  
 Out of the top 500 features 6 are present in query point

## 4.3.2. Without Class balancing

### 4.3.2.1. Hyper parameter tuning

In [0]:

```

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

```

```

rn.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))

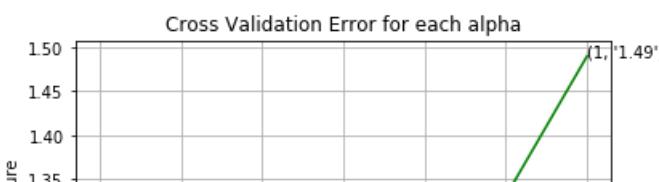
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.xscale('log')
plt.ylabel("Error measure")
plt.show()

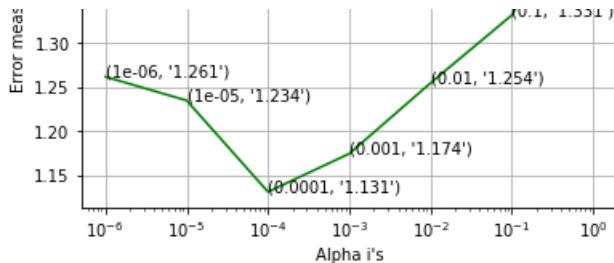
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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 : 1.261281096160343
for alpha = 1e-05
Log Loss : 1.2343951655207468
for alpha = 0.0001
Log Loss : 1.1308465079660144
for alpha = 0.001
Log Loss : 1.1740978833381044
for alpha = 0.01
Log Loss : 1.2541746076280738
for alpha = 0.1
Log Loss : 1.3311489140211308
for alpha = 1
Log Loss : 1.489995003000789

```





For values of best alpha = 0.0001 The train log loss is: 0.4338292680557092  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.1308465079660144  
 For values of best alpha = 0.0001 The test log loss is: 1.08676721073157

#### 4.3.2.2. Testing model with best hyper parameters

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
#
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=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.

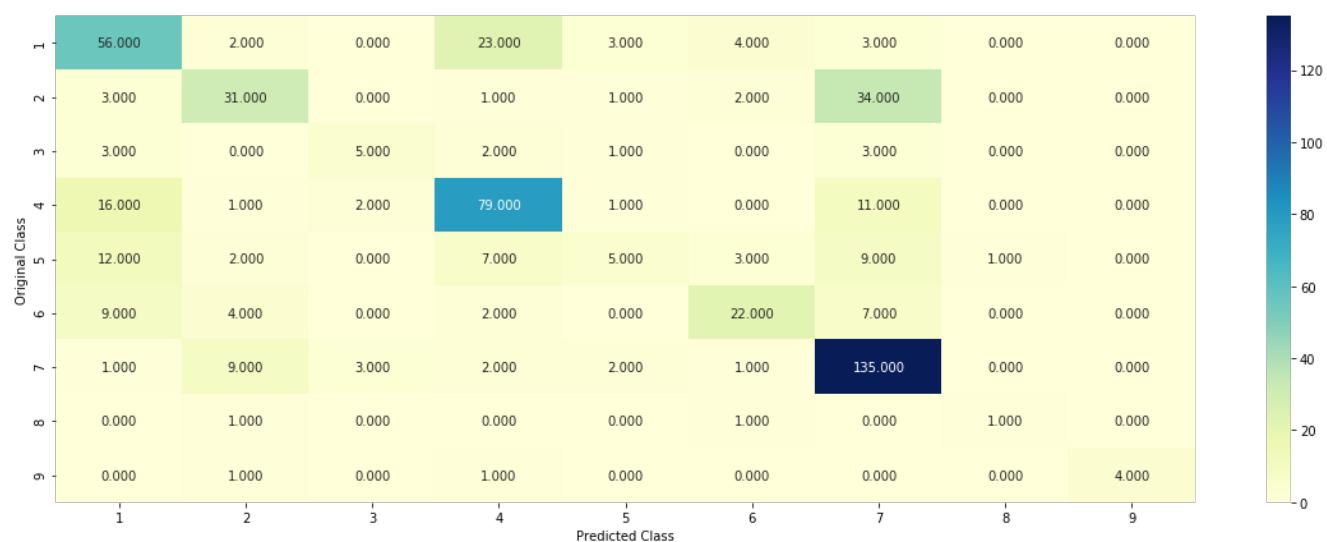
#
# video link:
#-----
```

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)

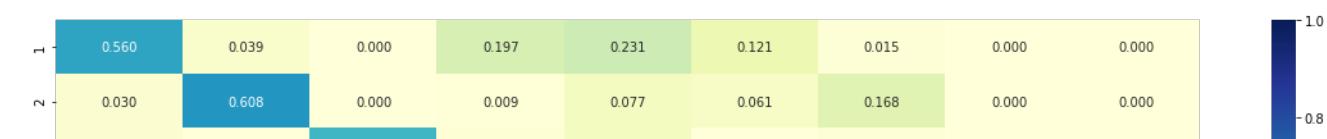
Log loss : 1.1308465079660144

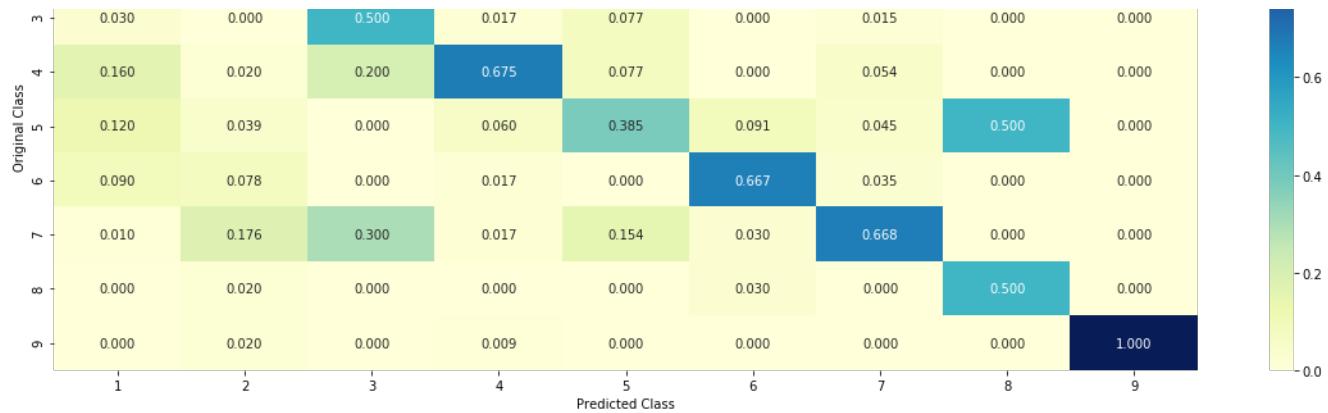
Number of mis-classified points : 0.36466165413533835

----- Confusion matrix -----

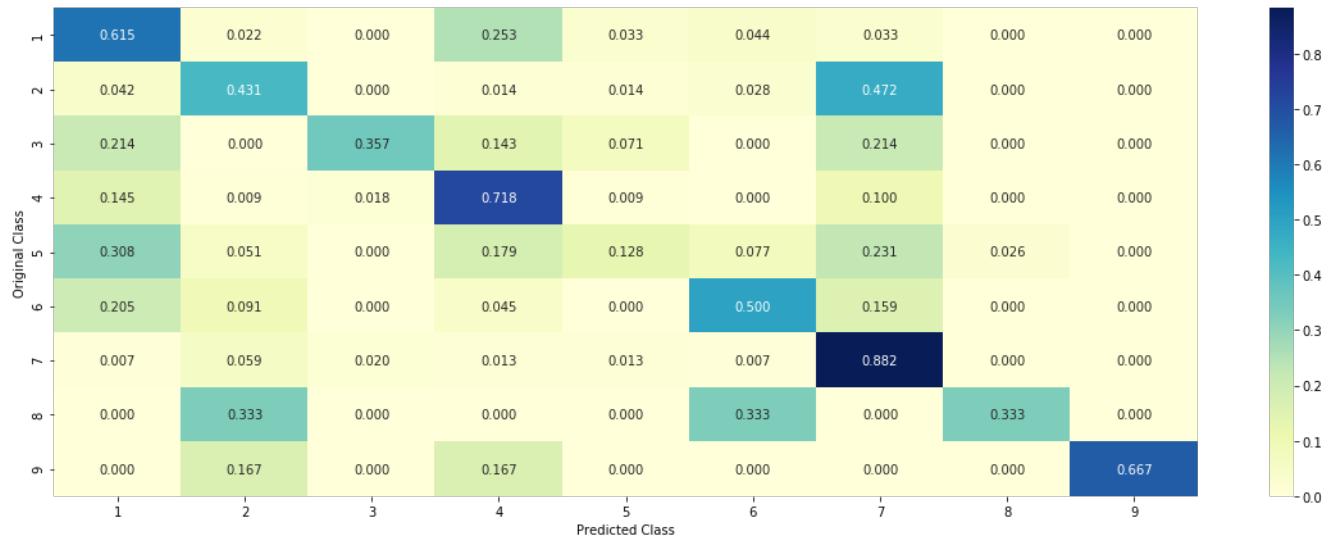


----- Precision matrix (Column Sum=1) -----





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



#### 4.3.2.3. Feature Importance, Correctly Classified point

In [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-" * 50)
get_imptfeature_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: [[0.1051 0.2525 0.0327 0.1287 0.0668 0.0731 0.3116 0.0091 0.0202]]  
Actual Class : 2  
-----  
286 Text feature [events] present in test data point [True]  
Out of the top 500 features 1 are present in query point

#### 4.3.2.4. Feature Importance, Incorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
```

```

predicted_cls = np.argmax(np.exp(np.log1p(sig_clf.coef_))), predicted_cls[0])
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(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_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: [[0.6872 0.0138 0.0008 0.1116 0.0213 0.1492 0.0095 0.0057 0.0009]]
Actual Class : 1
-----
Out of the top 500 features 0 are present in query point

```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper parameter tuning

In [0]:

```

# read more about support vector machines with linear kernels 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/
# -----


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


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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):

```

```

    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

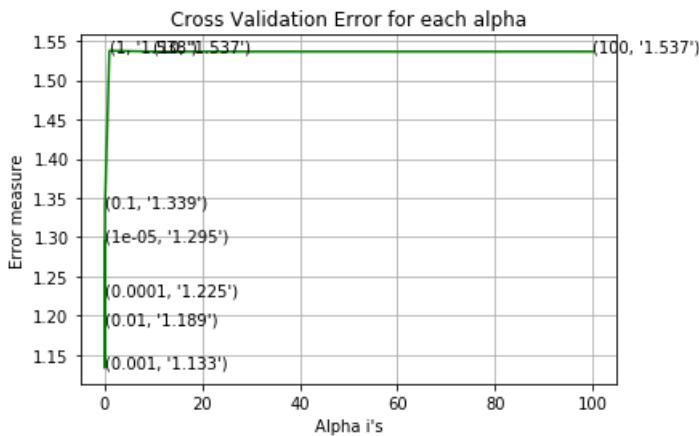
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "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 C = 1e-05
Log Loss : 1.2947751343886555
for C = 0.0001
Log Loss : 1.224917285000599
for C = 0.001
Log Loss : 1.1331895872345978
for C = 0.01
Log Loss : 1.1893344336839604
for C = 0.1
Log Loss : 1.3388346190621898
for C = 1
Log Loss : 1.5376547547575687
for C = 10
Log Loss : 1.536595444448099
for C = 100
Log Loss : 1.5365954814224203

```



```

For values of best alpha =  0.001 The train log loss is: 0.5133777904007458
For values of best alpha =  0.001 The cross validation log loss is: 1.1331895872345978
For values of best alpha =  0.001 The test log loss is: 1.1090027406074319

```

#### 4.4.2. Testing model with best hyper parameters

In [0]:

```

# read more about support vector machines with linear kernels 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)

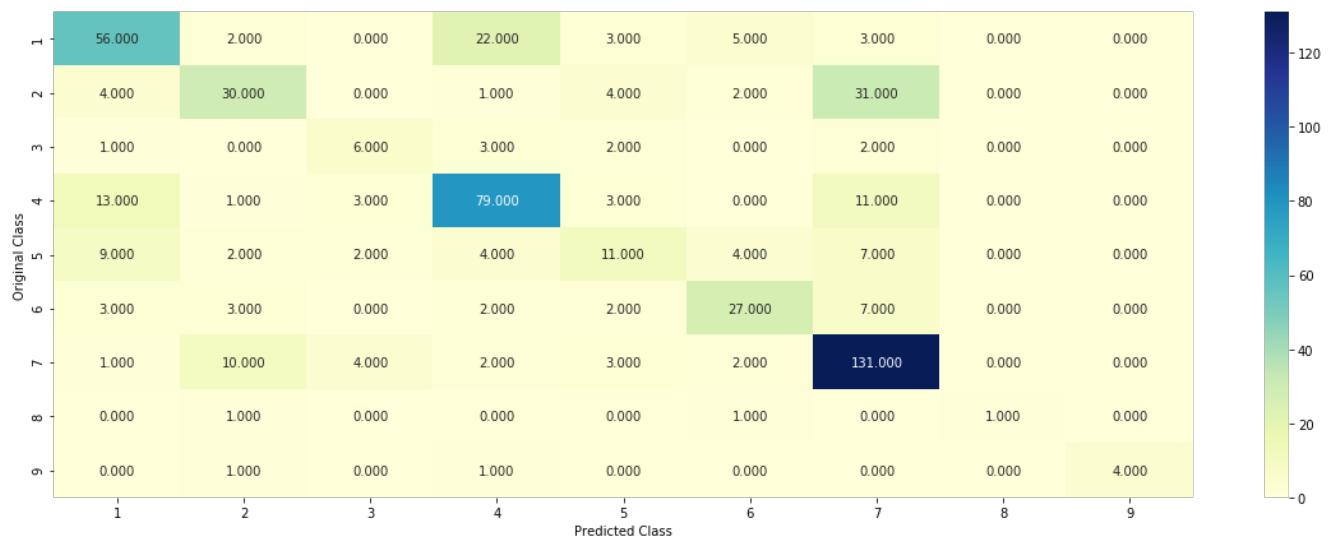
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----
```

# `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)`

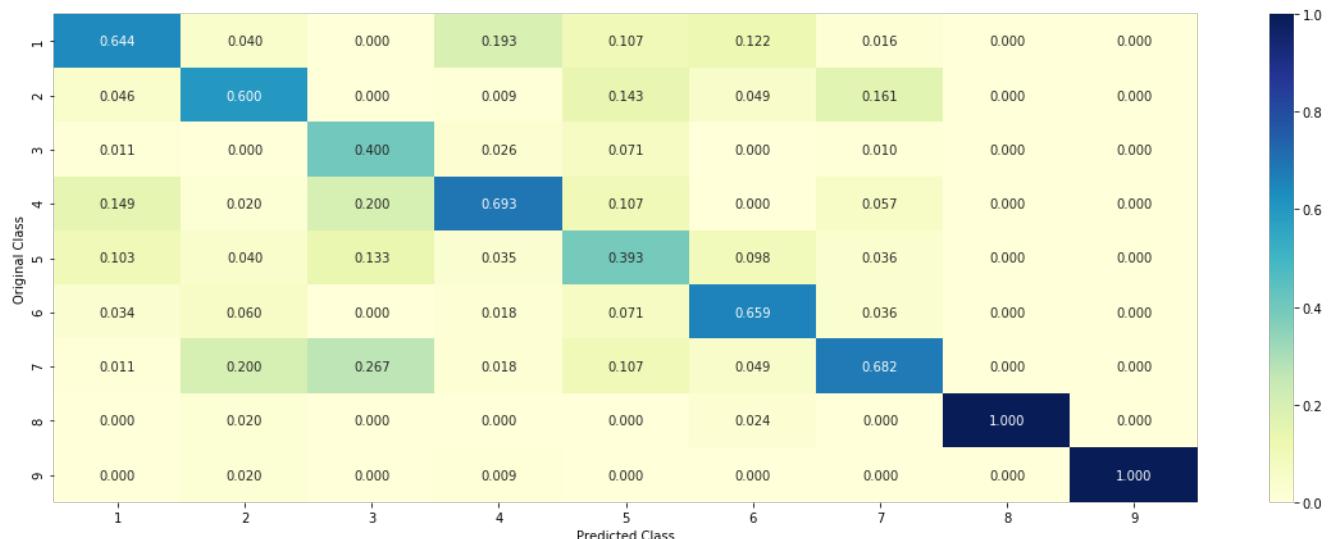
Log loss : 1.1331895872345978

Number of mis-classified points : 0.35150375939849626

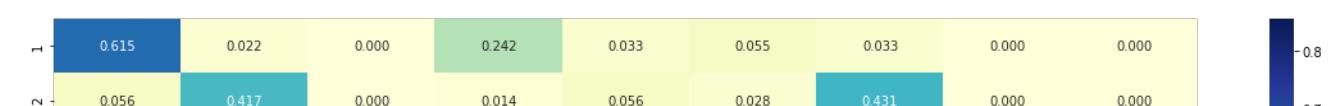
----- Confusion matrix -----

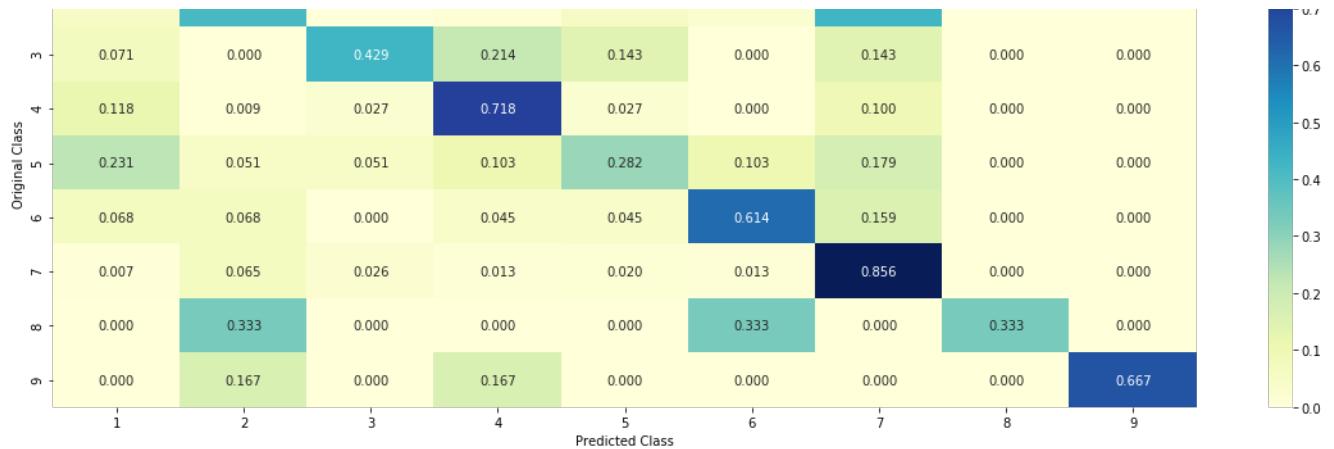


----- Precision matrix (Column Sum=1) -----



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





### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

In [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_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: [[0.1149 0.2478 0.0229 0.1356 0.0643 0.0701 0.3289 0.0055 0.0101]]  
Actual Class : 2

-----  
Out of the top 500 features 0 are present in query point

#### 4.3.3.2. For Incorrectly classified point

In [0]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_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: [[0.6815 0.0402 0.0029 0.0609 0.0111 0.1427 0.0511 0.0056 0.004 ]]  
Actual Class : 1

-----  
Out of the top 500 features 0 are present in query point

## 4.5 Random Forest Classifier

#### 4.5.1. Hyper parameter tuning (With One hot Encoding)

In [0]:

```
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample_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/
# -----


# 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 : 1.307623312567576
for n_estimators = 100 and max depth = 10
Log Loss : 1.2103331858346256
for n_estimators = 200 and max depth = 5
Log Loss : 1.2928801910119847
for n_estimators = 200 and max depth = 10
Log Loss : 1.2041456611117891
for n_estimators = 500 and max depth = 5
Log Loss : 1.2873062604994958
for n_estimators = 500 and max depth = 10
Log Loss : 1.2005986620063729
for n_estimators = 1000 and max depth = 5
Log Loss : 1.288704469712774
for n_estimators = 1000 and max depth = 10
Log Loss : 1.200209640206753
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2867441823026244
for n_estimators = 2000 and max depth = 10
Log Loss : 1.1993210520715294
For values of best estimator = 2000 The train log loss is: 0.6499875496837514
For values of best estimator = 2000 The cross validation log loss is: 1.1993210520715294
For values of best estimator = 2000 The test log loss is: 1.131290201198679

```

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

In [0]:

```

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

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

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

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

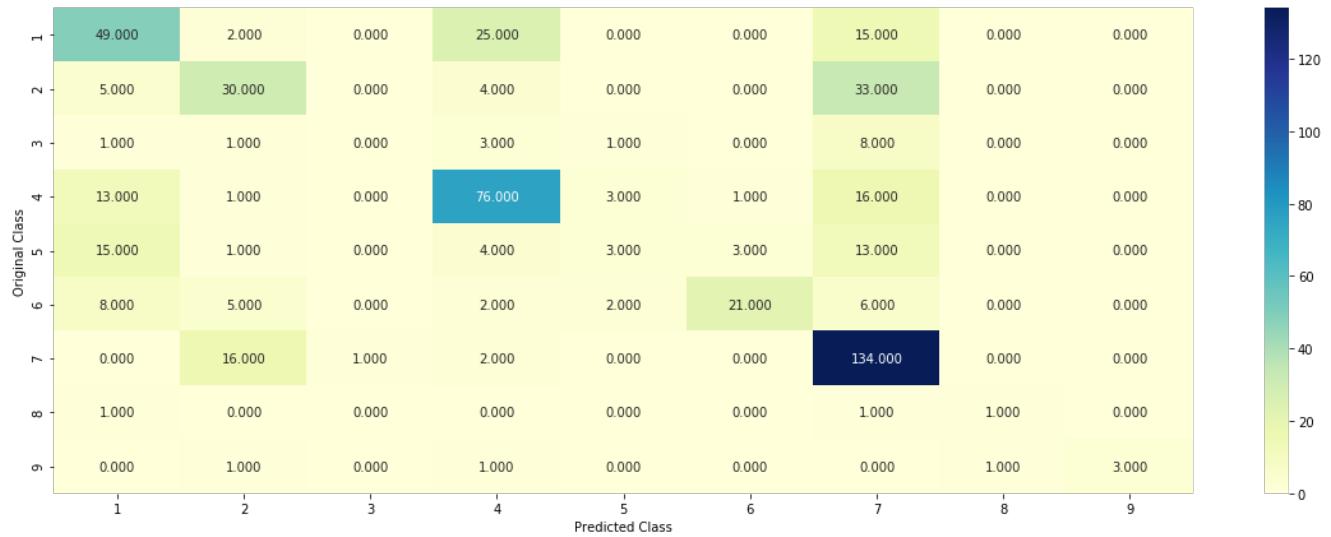

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha/2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

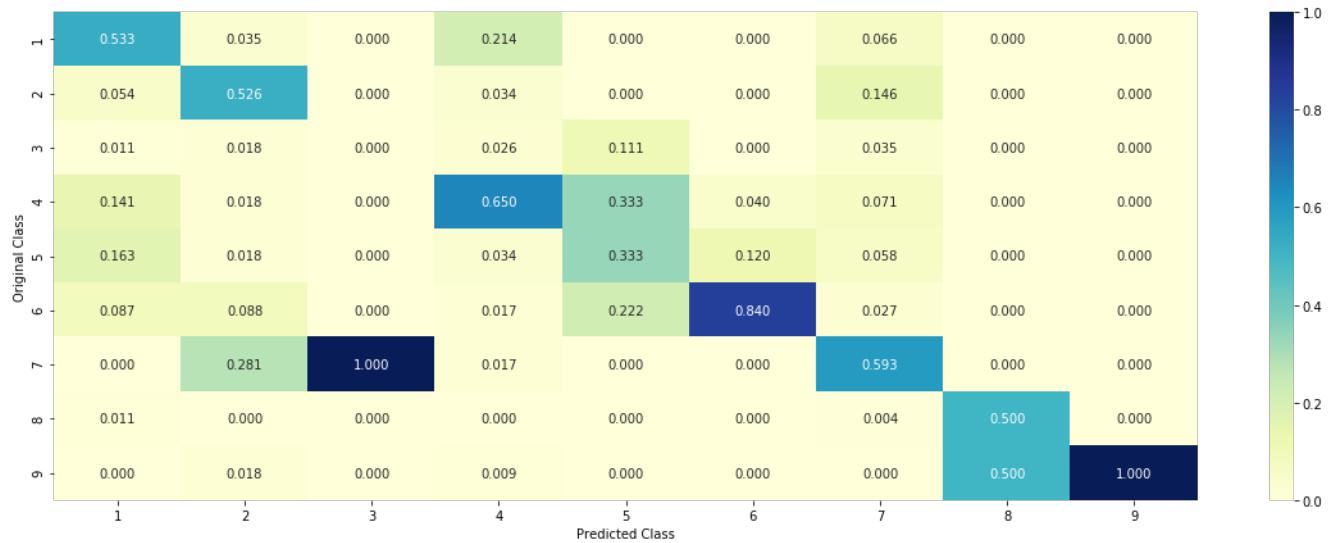
```

Log loss : 1.1993210520715294
Number of mis-classified points : 0.4041353383458647
----- Confusion matrix -----

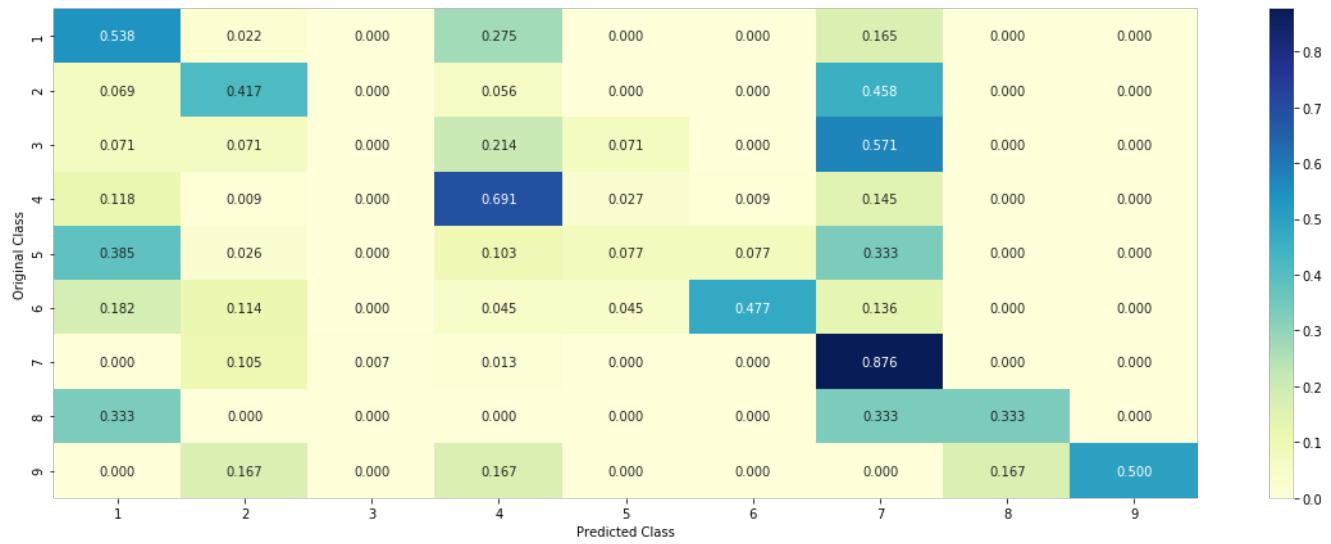
```



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



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



## 4.5.3. Feature Importance

### 4.5.3.1. Correctly Classified point

In [0]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_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 : 4
Predicted Class Probabilities: [[0.2302 0.1727 0.0241 0.2894 0.0676 0.0624 0.1314 0.0109 0.0113]]
Actual Class : 2
-----
0 Text feature [kinase] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
7 Text feature [oncogenic] present in test data point [True]
14 Text feature [receptor] present in test data point [True]
25 Text feature [growth] present in test data point [True]
37 Text feature [proliferation] present in test data point [True]
64 Text feature [potential] present in test data point [True]
Out of the top 100 features 7 are present in query point
```

#### 4.5.3.2. Incorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_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 : 6
Predicted Class Probabilities: [[0.3344 0.0222 0.0146 0.0708 0.0487 0.479 0.0222 0.0034 0.0048]]
Actual Class : 1
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
15 Text feature [suppressor] present in test data point [True]
17 Text feature [missense] present in test data point [True]
20 Text feature [loss] present in test data point [True]
22 Text feature [function] present in test data point [True]
33 Text feature [stability] present in test data point [True]
38 Text feature [activate] present in test data point [True]
42 Text feature [brcal] present in test data point [True]
43 Text feature [functional] present in test data point [True]
50 Text feature [yeast] present in test data point [True]
54 Text feature [cells] present in test data point [True]
57 Text feature [protein] present in test data point [True]
60 Text feature [deleterious] present in test data point [True]
65 Text feature [variants] present in test data point [True]
69 Text feature [proteins] present in test data point [True]
75 Text feature [functional] present in test data point [True]
```

```
'5 Text feature [ligana] present in test data point [True]
79 Text feature [ability] present in test data point [True]
88 Text feature [predicted] present in test data point [True]
99 Text feature [repair] present in test data point [True]
Out of the top 100 features 22 are present in query point
```

#### 4.5.3. Hyper parameter tuning (With Response Coding)

In [0]:

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

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

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

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


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


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

```

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

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

```

```

for n_estimators = 10 and max depth = 2
Log Loss : 2.1055621373146898
for n_estimators = 10 and max depth = 3
Log Loss : 1.6015202697918314
for n_estimators = 10 and max depth = 5
Log Loss : 1.4912413138500569
for n_estimators = 10 and max depth = 10
Log Loss : 1.7280136363330205
for n_estimators = 50 and max depth = 2
Log Loss : 1.6256523710655668
for n_estimators = 50 and max depth = 3
Log Loss : 1.371589882555836
for n_estimators = 50 and max depth = 5
Log Loss : 1.3764413116051462
for n_estimators = 50 and max depth = 10
Log Loss : 1.5830970136991327
for n_estimators = 100 and max depth = 2
Log Loss : 1.4825121292017371
for n_estimators = 100 and max depth = 3
Log Loss : 1.374907397395781
for n_estimators = 100 and max depth = 5
Log Loss : 1.3785244548859212
for n_estimators = 100 and max depth = 10
Log Loss : 1.6119745178536917
for n_estimators = 200 and max depth = 2
Log Loss : 1.5255495186393262
for n_estimators = 200 and max depth = 3
Log Loss : 1.3839193095819002
for n_estimators = 200 and max depth = 5
Log Loss : 1.3835174192374495
for n_estimators = 200 and max depth = 10
Log Loss : 1.6343972175560002
for n_estimators = 500 and max depth = 2
Log Loss : 1.575025382974805
for n_estimators = 500 and max depth = 3
Log Loss : 1.4382079910063021
for n_estimators = 500 and max depth = 5
Log Loss : 1.371863723003668
for n_estimators = 500 and max depth = 10
Log Loss : 1.6116757755476054
for n_estimators = 1000 and max depth = 2
Log Loss : 1.5574666129560928
for n_estimators = 1000 and max depth = 3
Log Loss : 1.4551115317205892
for n_estimators = 1000 and max depth = 5
Log Loss : 1.3484702370341621
for n_estimators = 1000 and max depth = 10
Log Loss : 1.6236239757239408
For values of best alpha = 1000 The train log loss is: 0.06358977495638705
For values of best alpha = 1000 The cross validation log loss is: 1.3484702370341621
For values of best alpha = 1000 The test log loss is: 1.404667852894824

```

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

In [0]:

```
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample_size=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)
```

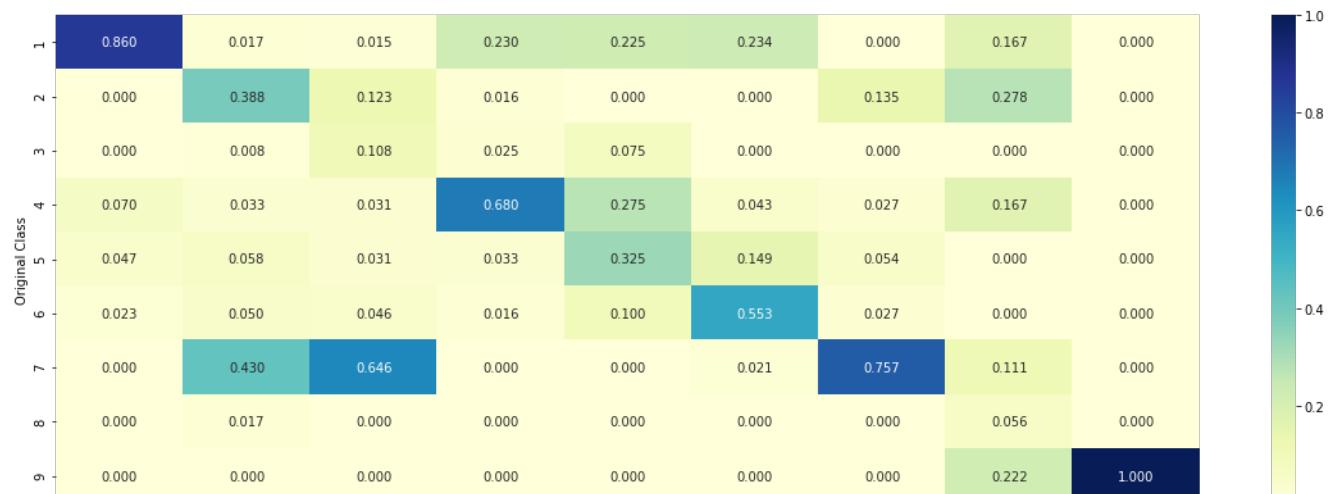
Log loss : 1.3484702370341621

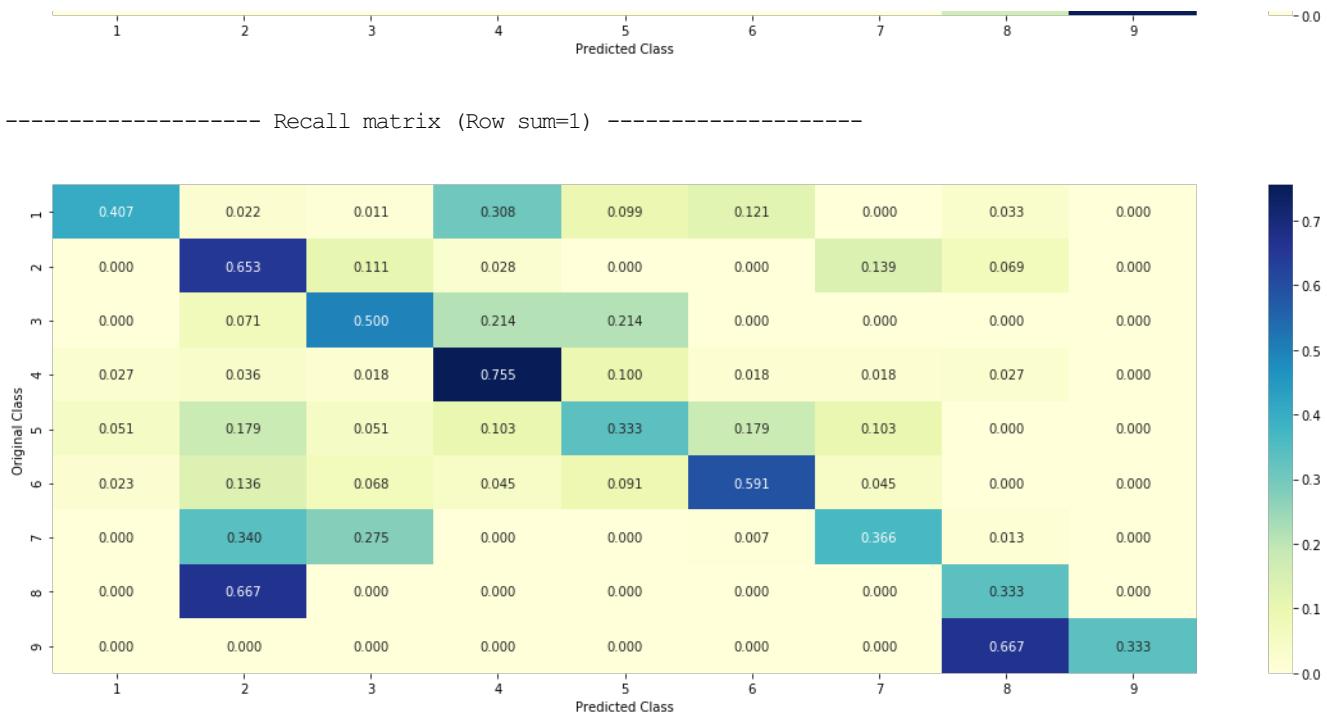
Number of mis-classified points : 0.48872180451127817

----- Confusion matrix -----



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





## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point

In [0]:

```

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

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

```

Predicted Class : 2  
Predicted Class Probabilities: [[0.0171 0.6042 0.0813 0.0192 0.0251 0.0446 0.1647 0.0317 0.0121]]  
Actual Class : 2

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

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

#### 4.5.5.2. Incorrectly Classified point

In [0]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.14    0.014   0.1166  0.1227  0.2904  0.2755  0.0101  0.0144  0.0164]]
Actual Class : 1
-----
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

## 4.7 Stack the models

## 4.7.1 testing with hyper parameter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, 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-intuiti
on-1/
#-----

# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modul
es/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-deri
vation-copy-8/
# -----


# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modul
es/generated/sklearn.ensemble.RandomForestClassifier.html
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impu
rity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_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)
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_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.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: 1.14  
Support vector machines : Log Loss: 1.5  
Naive Bayes : Log Loss: 1.29

```
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 1.817  
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 1.714  
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.308  
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.180  
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.428  
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.794
```

#### 4.7.2 testing the model with the best hyper parameters

In [0]:

```

lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=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)

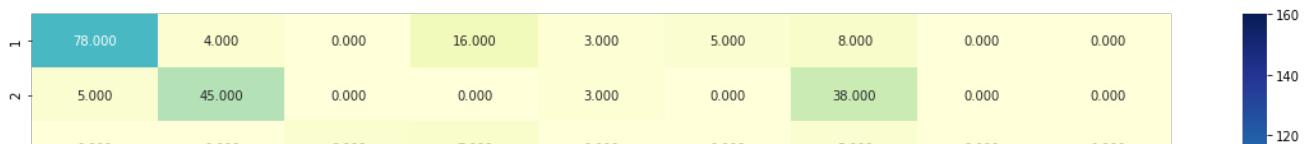
log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

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))

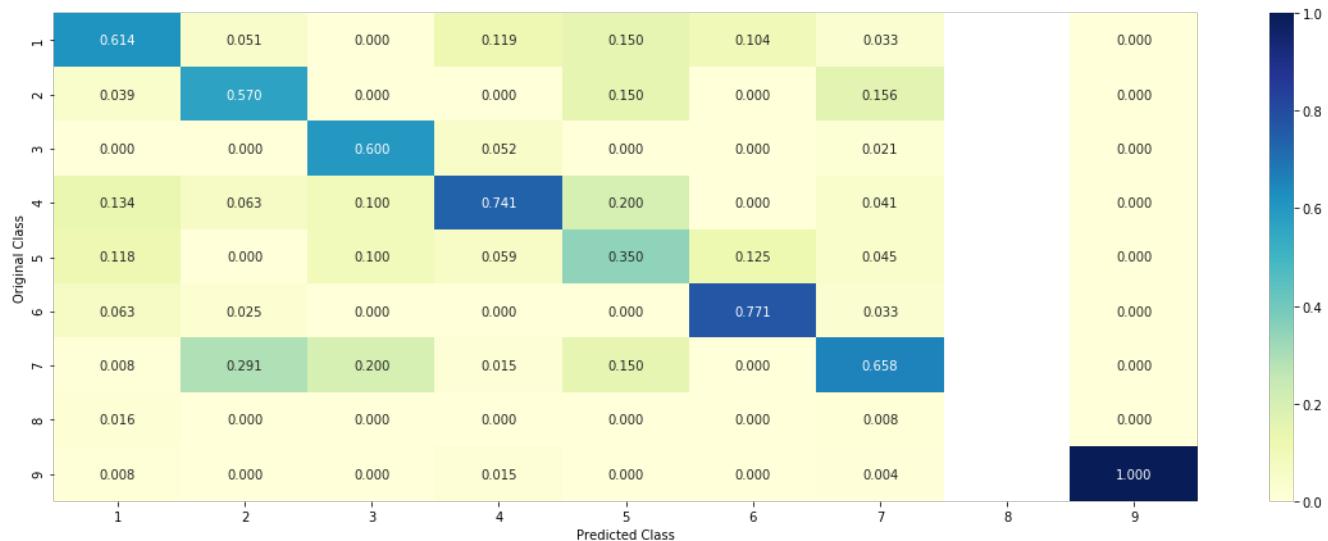
```

```
Log loss (train) on the stacking classifier : 0.48590099859011093  
Log loss (CV) on the stacking classifier : 1.1802005856853464  
Log loss (test) on the stacking classifier : 1.1474111907767106  
Number of missclassified point : 0.3443609022556391  
Confusion matrix:
```

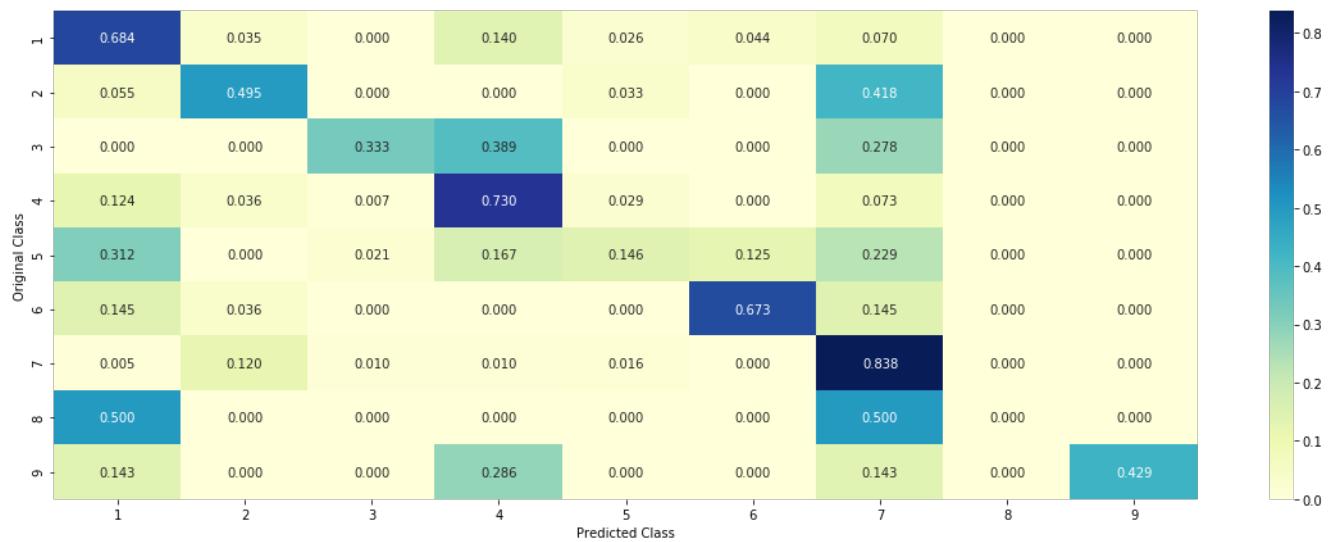




----- Precision matrix (Column Sum=1) -----



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



#### 4.7.3 Maximum Voting classifier

In [0]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
```

```

Coding)))
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))

```

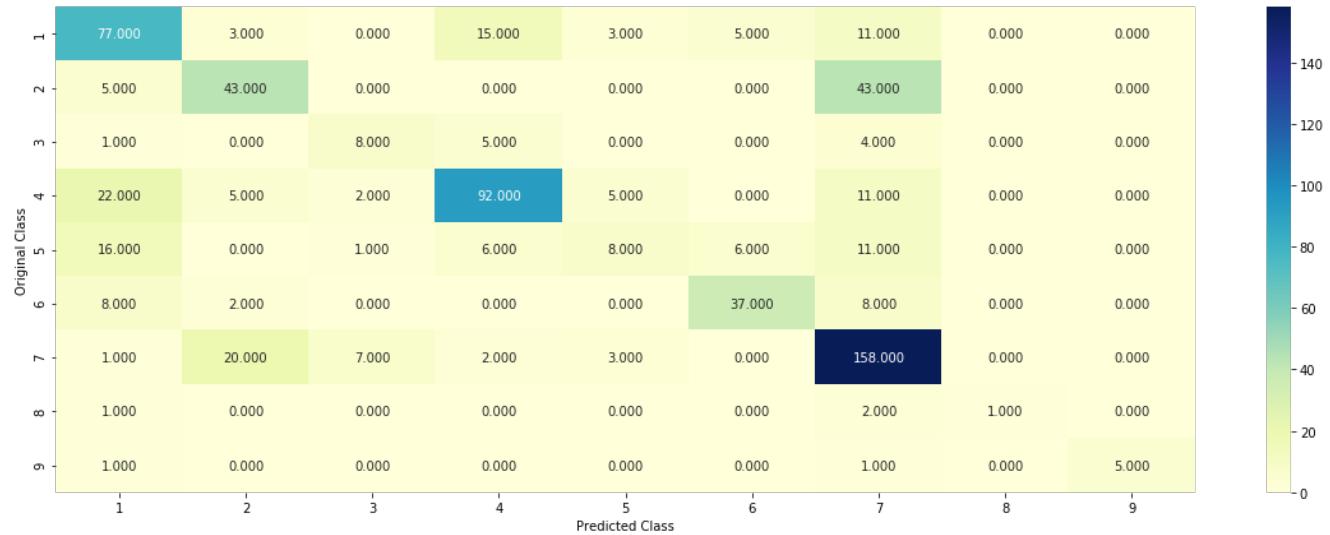
Log loss (train) on the VotingClassifier : 0.8020668588875274

Log loss (CV) on the VotingClassifier : 1.1738959037960004

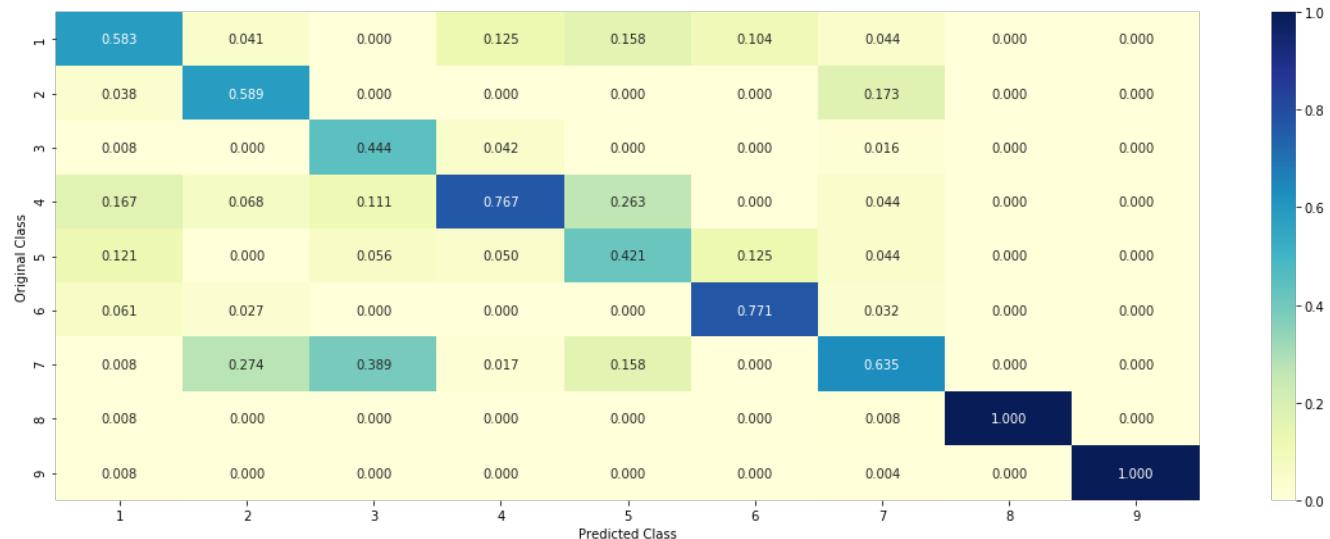
Log loss (test) on the VotingClassifier : 1.1263883270461903

Number of missclassified point : 0.3548872180451128

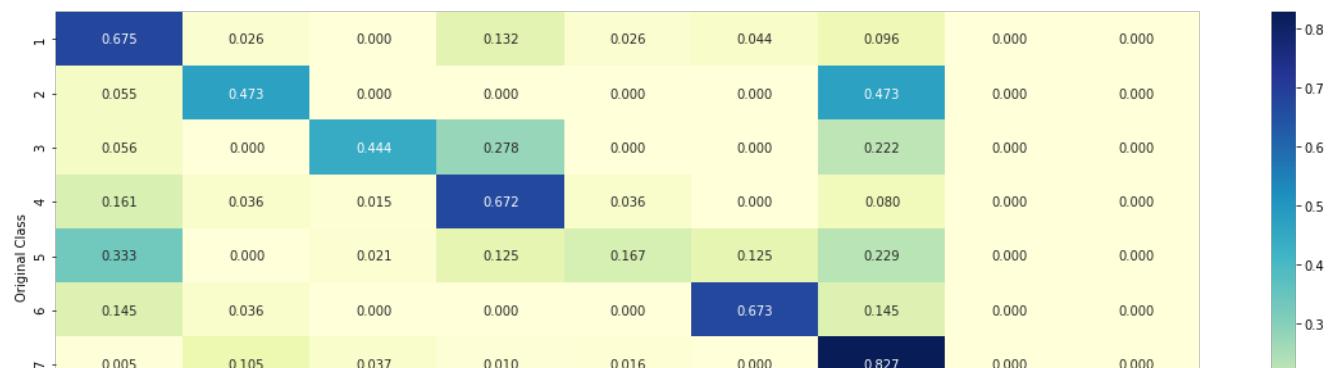
----- Confusion matrix -----

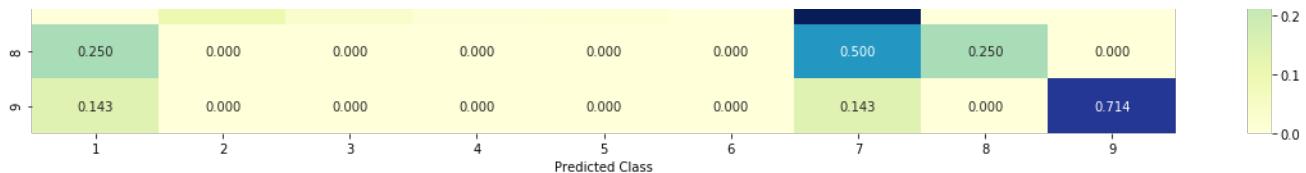


----- Precision matrix (Column Sum=1) -----



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





## 5. Assignment

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

### b) Using top 1000 words based of tf-idf values

In [0]:

```
# https://stackoverflow.com/questions/51655598/how-to-select-top-1000-words-using-tf-idf-vector

text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])

# 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)

print(train_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)

(2124, 1000)
(532, 1000)
(665, 1000)
```

In [0]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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:
```

```

#-----



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.classes_, eps=1e-15))

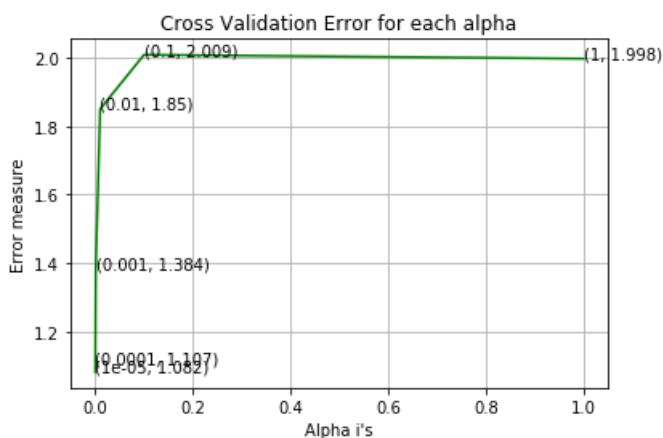
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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_loss(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, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.0816109798745372  
 For values of alpha = 0.0001 The log loss is: 1.107296590747186  
 For values of alpha = 0.001 The log loss is: 1.3840733930280775  
 For values of alpha = 0.01 The log loss is: 1.850412197384709  
 For values of alpha = 0.1 The log loss is: 2.0092080835603645  
 For values of alpha = 1 The log loss is: 1.9977082199934917



For values of best alpha = 1e-05 The train log loss is: 0.6799418356387261  
 For values of best alpha = 1e-05 The cross validation log loss is: 1.0816109798745372  
 For values of best alpha = 1e-05 The test log loss is: 1.1516384548684324

## Machine Learning Models

In [0]:

```
#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 will 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 [0]:

```
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 [0]:

```
# 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_imptfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

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

```
# 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]]
from scipy.sparse import hstack
from scipy import sparse

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)).tocsr()
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 = hstack((sparse.csr_matrix(train_gene_var_responseCoding), train_text_feature_onehotCoding)).tocsr()
# test_x_responseCoding = hstack((sparse.csr_matrix(test_gene_var_responseCoding), test_text_feature_onehotCoding)).tocsr()
# cv_x_responseCoding = hstack((sparse.csr_matrix(cv_gene_var_responseCoding), cv_text_feature_onehotCoding)).tocsr()

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

In [0]:

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

One hot encoding features :

```
(number of data points * number of features) in train data = (2124, 3195)
(number of data points * number of features) in test data = (665, 3195)
(number of data points * number of features) in cross validation data = (532, 3195)
```

In [0]:

```
print(" Response encoding features :")
print("number of data points * number of features) in train data = ", train_x_responseCoding.shape)
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 test data = (500, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

## 4.1. Base Line Model

### 4.1.1. Naive Bayes

#### 4.1.1.1. Hyper parameter tuning

In [0]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```

sig_clf.fit(train_x_onehotCoding, train_y)

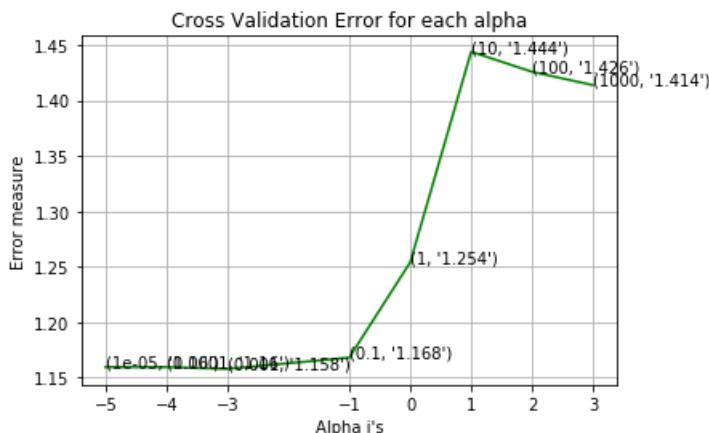
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_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-05
Log Loss : 1.1596148026937148
for alpha = 0.0001
Log Loss : 1.1595503925756034
for alpha = 0.001
Log Loss : 1.1579803572412544
for alpha = 0.01
Log Loss : 1.1680851520537079
for alpha = 0.1
Log Loss : 1.253970996992324
for alpha = 1
Log Loss : 1.443775574602402
for alpha = 10
Log Loss : 1.425667301618946
for alpha = 100
Log Loss : 1.4136742919374696

```



```

For values of best alpha = 0.001 The train log loss is: 0.4499189910359893
For values of best alpha = 0.001 The cross validation log loss is: 1.1579803572412544
For values of best alpha = 0.001 The test log loss is: 1.2246715449568282

```

#### 4.1.1.2. Testing the model with best hyper parameters

In [0]:

```

# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -----
# default parameters
# 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/sklear

```

```

rn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# 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
# -----

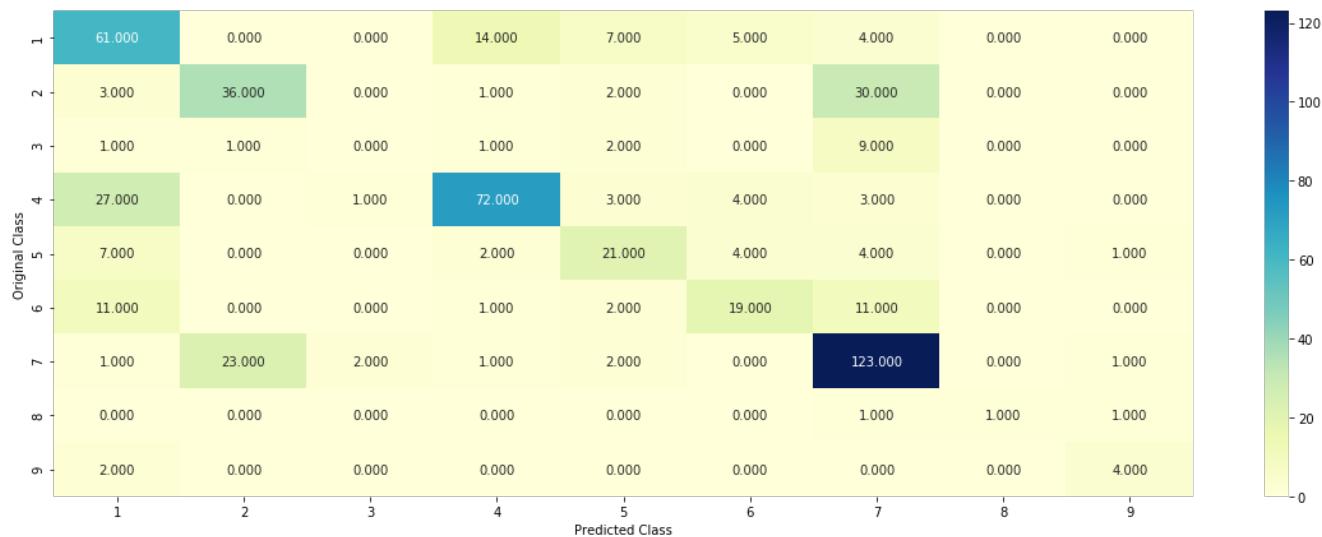
```

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 probabilitis 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())))

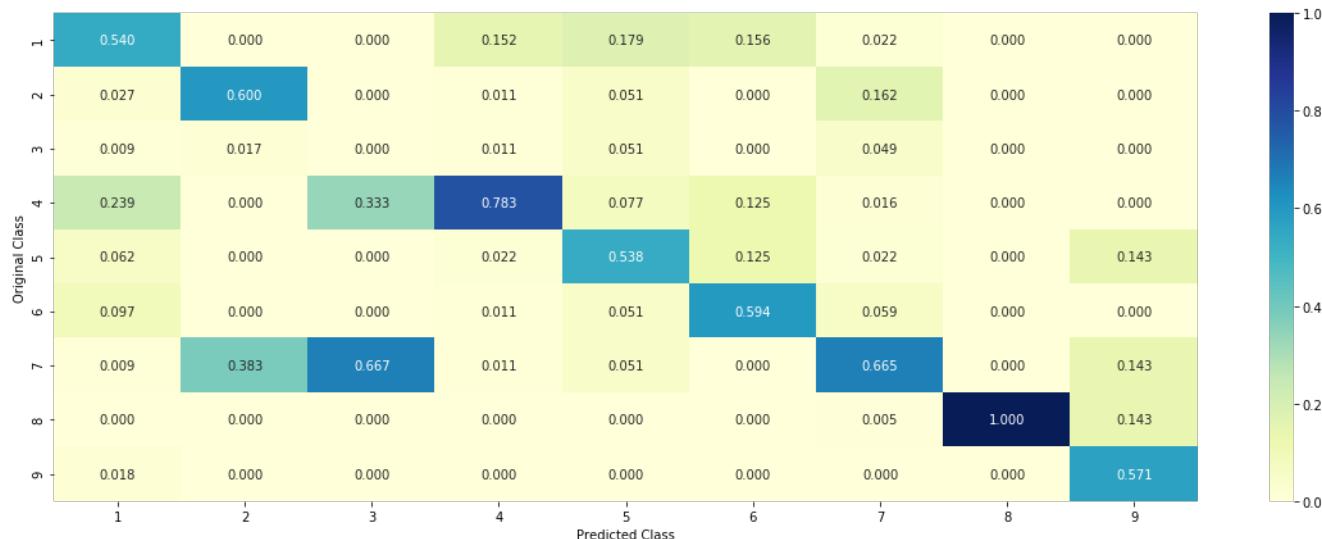
Log Loss : 1.1579803572412544

Number of missclassified point : 0.36654135338345867

----- Confusion matrix -----

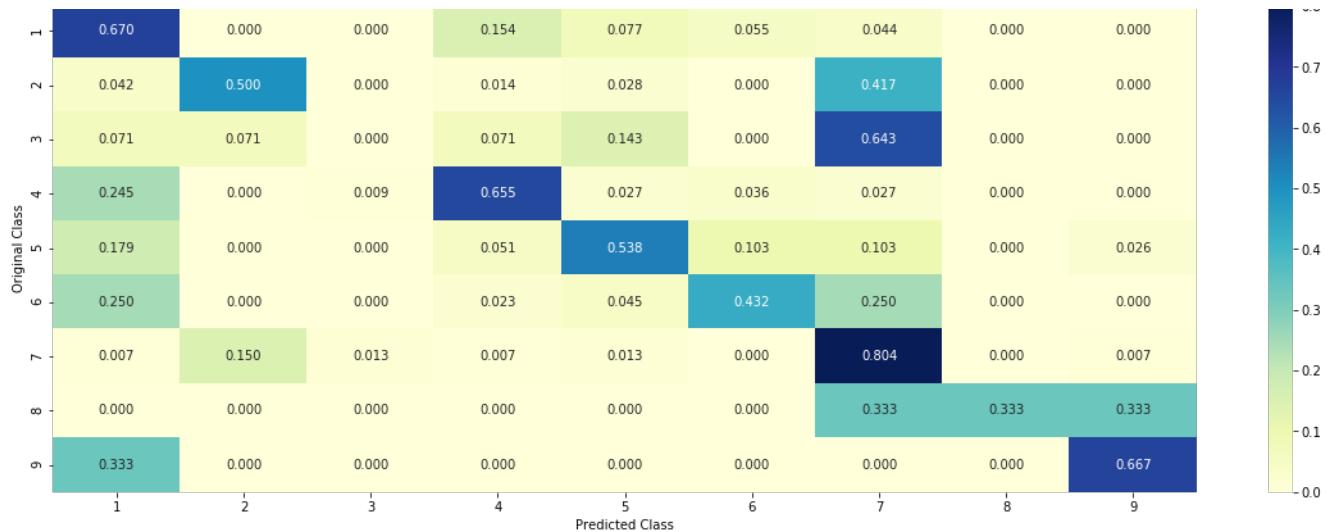


----- Precision matrix (Column Sum=1) -----



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

----- 0.8 -----



#### 4.1.1.3. Feature Importance, Correctly classified point

In [0]:

```
test_point_index = 12
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(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-" * 50)
get_imptfeature_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: [[0.0553 0.0544 0.01 0.0648 0.03 0.0323 0.7457 0.0046 0.0029]]  
Actual Class : 7

-----  
27 Text feature [05] present in test data point [True]  
51 Text feature [12] present in test data point [True]  
54 Text feature [113] present in test data point [True]  
65 Text feature [016] present in test data point [True]  
90 Text feature [108] present in test data point [True]  
Out of the top 100 features 5 are present in query point

#### 4.1.1.4. Feature Importance, Incorrectly classified point

In [0]:

```
test_point_index = 16
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(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-" * 50)
get_imptfeature_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 : 2  
Predicted Class Probabilities: [[0.075 0.531 0.0129 0.0833 0.1379 0.0415 0.109 0.0059 0.0037]]  
Actual Class : 1

-----  
46 Text feature [1000] present in test data point [True]  
81 Text feature [12] present in test data point [True]  
82 Text feature [113] present in test data point [True]  
Out of the top 100 features 3 are present in query point

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

In [0]:

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

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

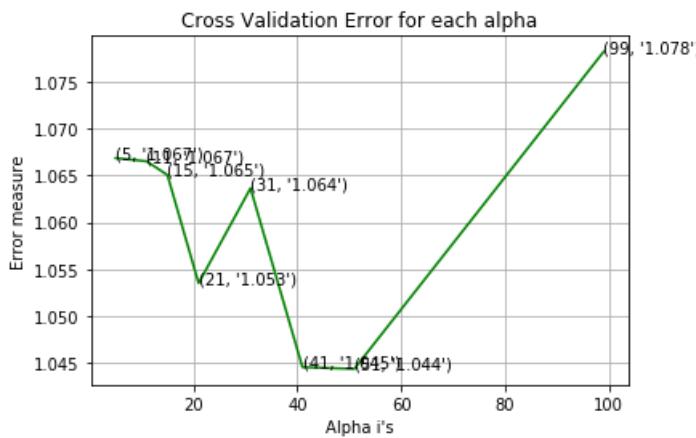
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
```

```

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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

for alpha = 5
Log Loss : 1.0668565188476375
for alpha = 11
Log Loss : 1.066514213183374
for alpha = 15
Log Loss : 1.0650487722032471
for alpha = 21
Log Loss : 1.053495292470407
for alpha = 31
Log Loss : 1.0636392529839451
for alpha = 41
Log Loss : 1.0445239392706949
for alpha = 51
Log Loss : 1.044344579540219
for alpha = 99
Log Loss : 1.078217337366669

```



```

For values of best alpha = 51 The train log loss is: 0.8714923977081717
For values of best alpha = 51 The cross validation log loss is: 1.044344579540219
For values of best alpha = 51 The test log loss is: 1.1181257112775533

```

#### 4.2.2. Testing the model with best hyper parameters

In [0]:

```

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

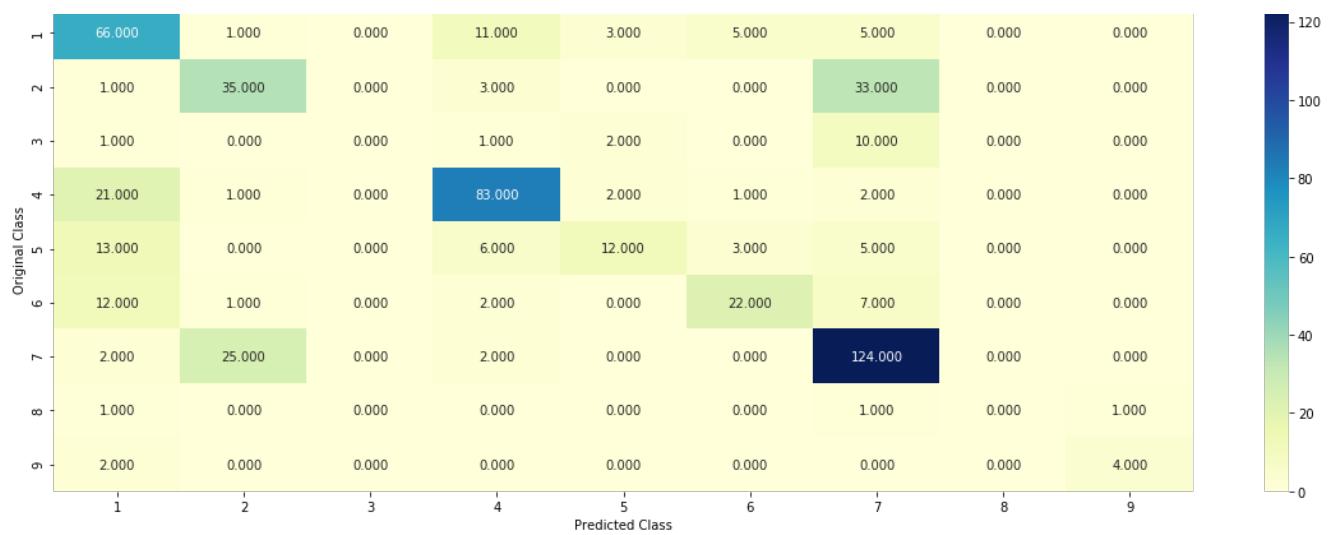
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X) : Predict the class labels for the provided data
# predict_proba(X) : Return probability estimates for the test data X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
# -----
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)

```

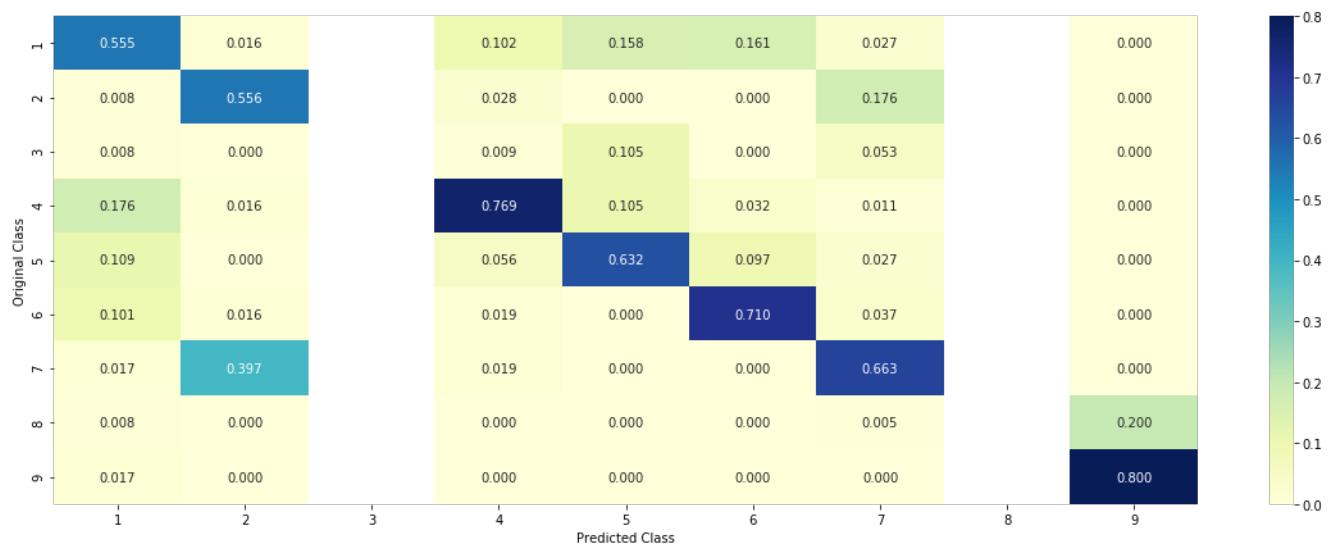
```

Log loss : 1.044344579540219
Number of mis-classified points : 0.34962406015037595
----- Confusion matrix -----

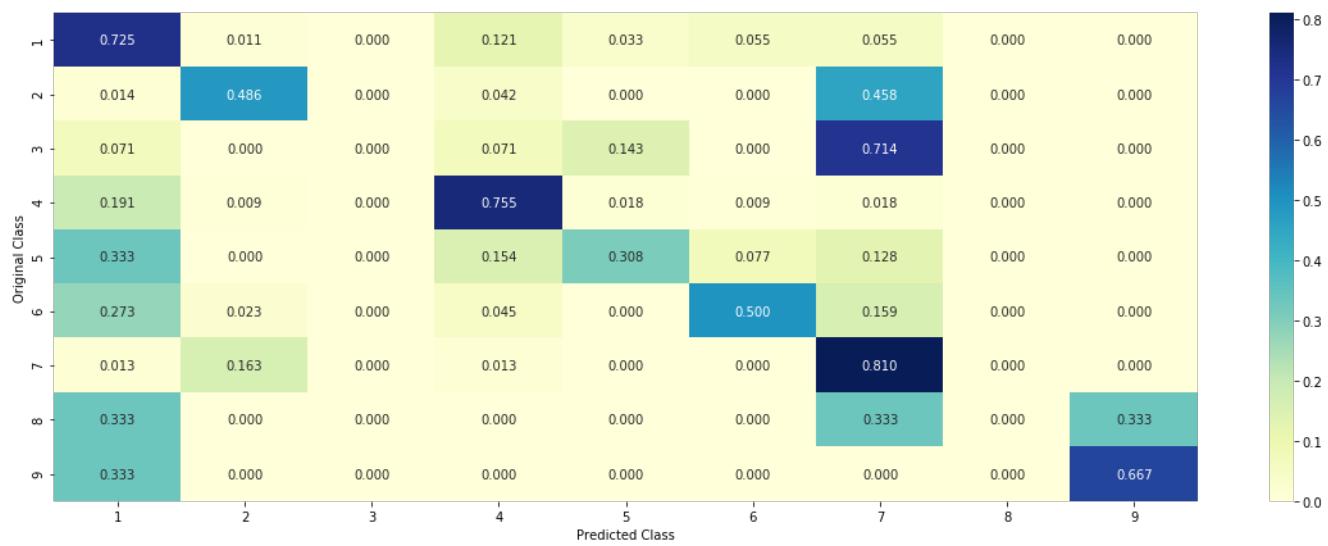
```



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



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



#### 4.2.3.Sample Query point -1

In [0]:

```
test_x_responseCoding[0].todense()
```

Out [0]:

```
matrix([[0.09565217, 0.15652174, 0.08695652, ... , 0.06626406, 0.010291863]])
```

In [0]:

```

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

test_point_index = 34
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 ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neig
hbors[1][0]])
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))

Predicted Class : 7
Actual Class : 7
The 51 nearest neighbours of the test points belongs to classes [7 7 7 2 7 6 7 7 7 2 2 7 2 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 2 7 6 2 2 7
7 7 7 7 7 7 7 7 8 7 7 7 7 7]
Frequency of nearest points : Counter({7: 41, 2: 7, 6: 2, 8: 1})

```

#### **4.2.3. Sample Query point -2**

In [0]:

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

test_point_index = 14
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 ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neig
hbors[1][0]])
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

### 4.3. Logistic Regression

#### 4.3.1. With Class balancing

#### 4.3.1.1 Hyper parameter tuning

Tn [0]:

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

```

power_etc.,
# 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, 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.xscale('log')
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', 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))

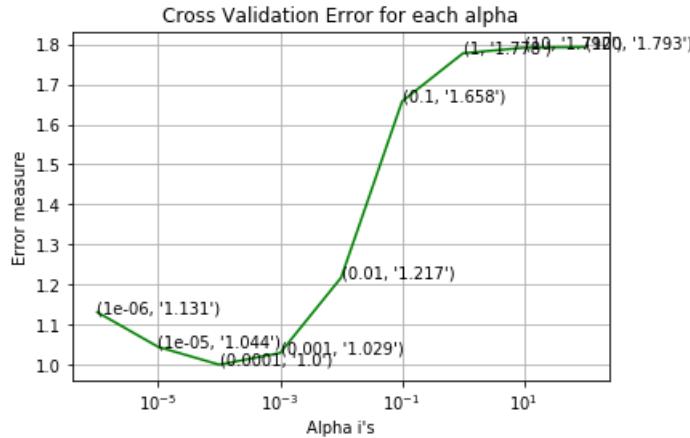
for alpha = 1e-06
Log Loss : 1.1308580852077583
for alpha = 1e-05
Log Loss : 1.0440609343118081
for alpha = 0.0001
Log Loss : 0.9997234705856081
for alpha = 0.001

```

```

Log Loss : 1.0289498852736472
for alpha = 0.01
Log Loss : 1.217253708218955
for alpha = 0.1
Log Loss : 1.6577045475713934
for alpha = 1
Log Loss : 1.7778910952392466
for alpha = 10
Log Loss : 1.7915438903497933
for alpha = 100
Log Loss : 1.793149774088052

```



For values of best alpha = 0.0001 The train log loss is: 0.39608112680093316  
 For values of best alpha = 0.0001 The cross validation log loss is: 0.9997234705856081  
 For values of best alpha = 0.0001 The test log loss is: 1.0340032175009046

#### 4.3.1.2. Testing the model with best hyper parameters

In [0]:

```

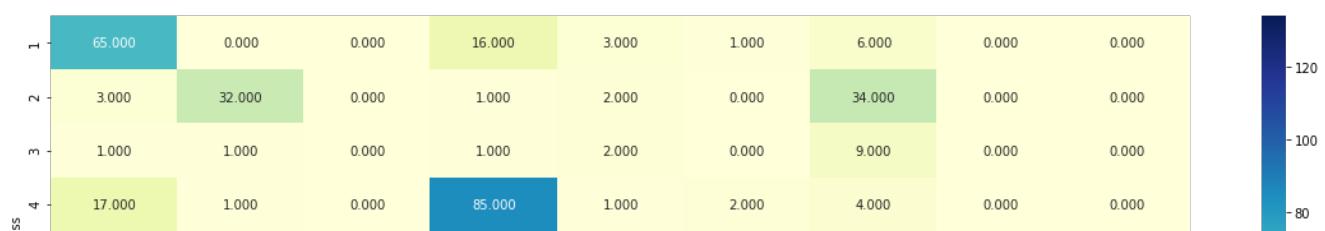
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None,
# tol=None,
# shuffle=True, verbose=0, 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)

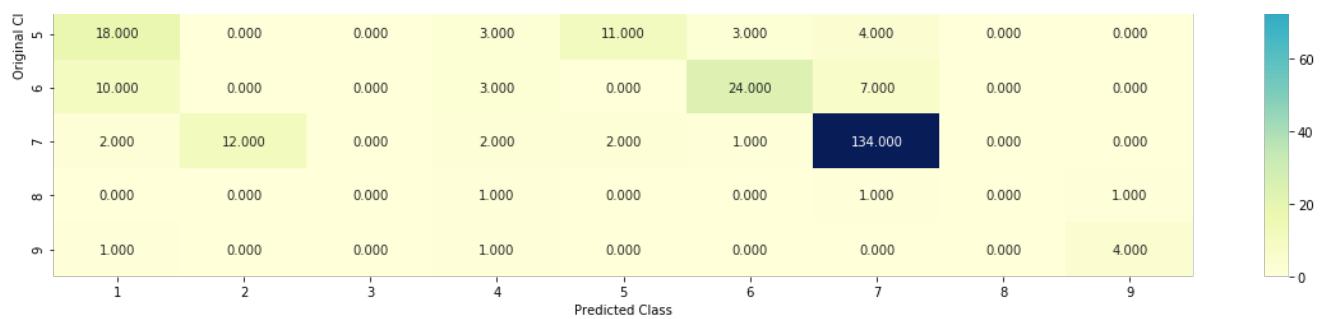
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

#-----
# 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)

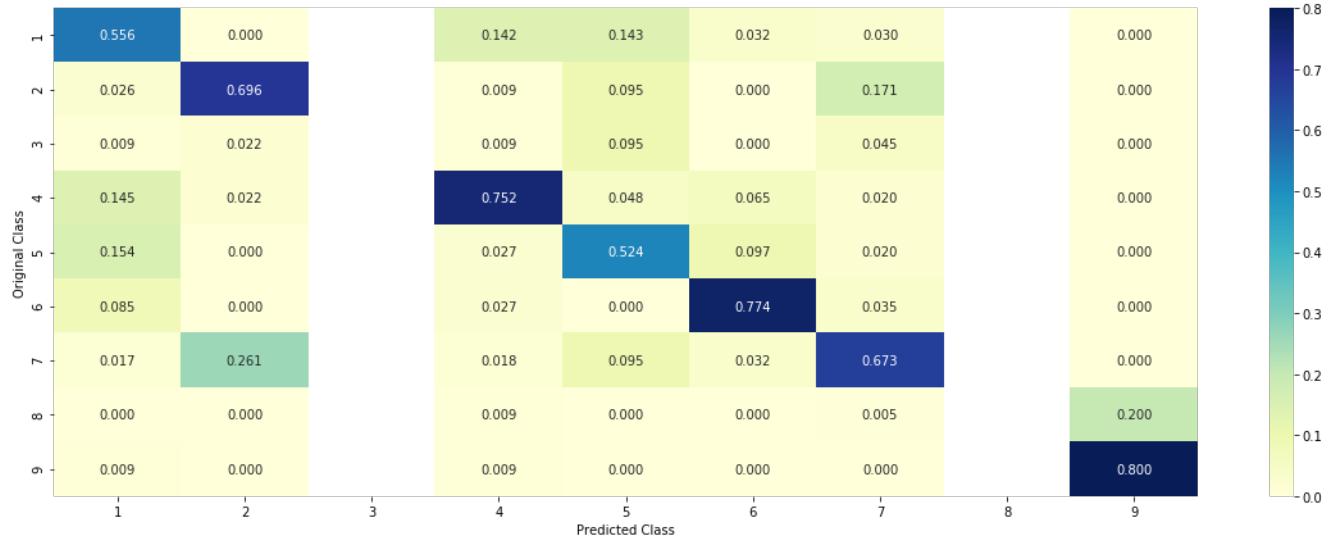
```

Log loss : 0.9997234705856081  
 Number of mis-classified points : 0.33270676691729323  
 ----- Confusion matrix -----

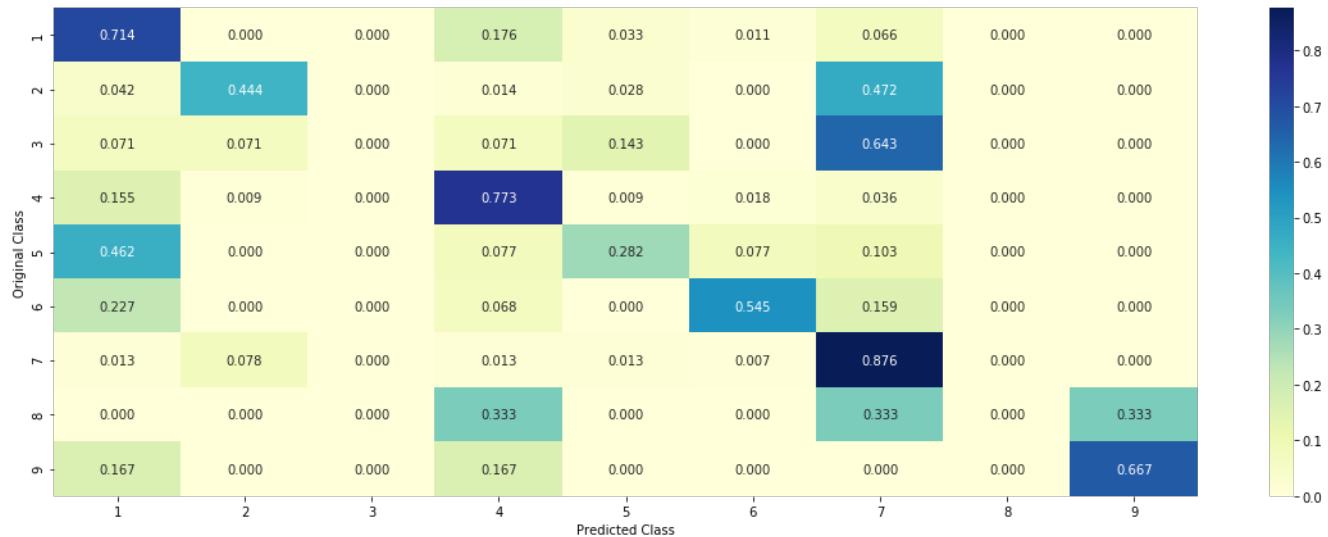




----- Precision matrix (Column Sum=1) -----



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



#### 4.3.1.3. Feature Importance

In [0]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)):
            tabulte_list.append([incresingorder_ind, "Other", "No"])
    return tabulte_list
```

```

++ (( < +) * + NOC IN removed_max, .
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 important 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[link text]

In [0]:

```

# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 12
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_i
ndex]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_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: [[7.400e-03 3.450e-02 1.400e-03 8.100e-03 5.300e-03 4.650e-02 8.853e-01
1.090e-02 6.000e-04]]
Actual Class : 7
-----
24 Text feature [120] present in test data point [True]
174 Text feature [1271] present in test data point [True]
339 Text feature [109] present in test data point [True]
364 Text feature [125] present in test data point [True]
458 Text feature [0065] present in test data point [True]
Out of the top 500 features 5 are present in query point

```

#### 4.3.1.3.1. Incorrectly Classified point

In [0]:

```

# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 19
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_i
ndex]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_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: [[0.1736 0.0539 0.0097 0.0912 0.6004 0.0538 0.0091 0.0059 0.0024]]
Actual Class : 1
-----
Out of the top 500 features 0 are present in query point

```

## 4.3.2. Without Class balancing

### 4.3.2.1. Hyper parameter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, 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-intuiti
on-1/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
rn.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.xscale('log')
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, pred
```

```

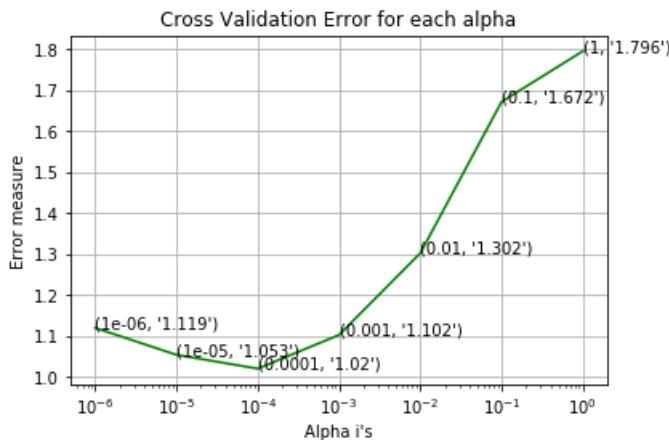
    predict_y, labels=clf.classes_, eps=1e-15)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
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 : 1.1191869784250215
for alpha = 1e-05
Log Loss : 1.0532451197643358
for alpha = 0.0001
Log Loss : 1.019533719050228
for alpha = 0.001
Log Loss : 1.1018788473501375
for alpha = 0.01
Log Loss : 1.3018964529795172
for alpha = 0.1
Log Loss : 1.6720752970763495
for alpha = 1
Log Loss : 1.7958066108826647

```



```

For values of best alpha =  0.0001 The train log loss is: 0.3877573243335947
For values of best alpha =  0.0001 The cross validation log loss is: 1.019533719050228
For values of best alpha =  0.0001 The test log loss is: 1.0490243461809086

```

#### 4.3.2.2. Testing model with best hyper parameters

In [0]:

```

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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:
#-----

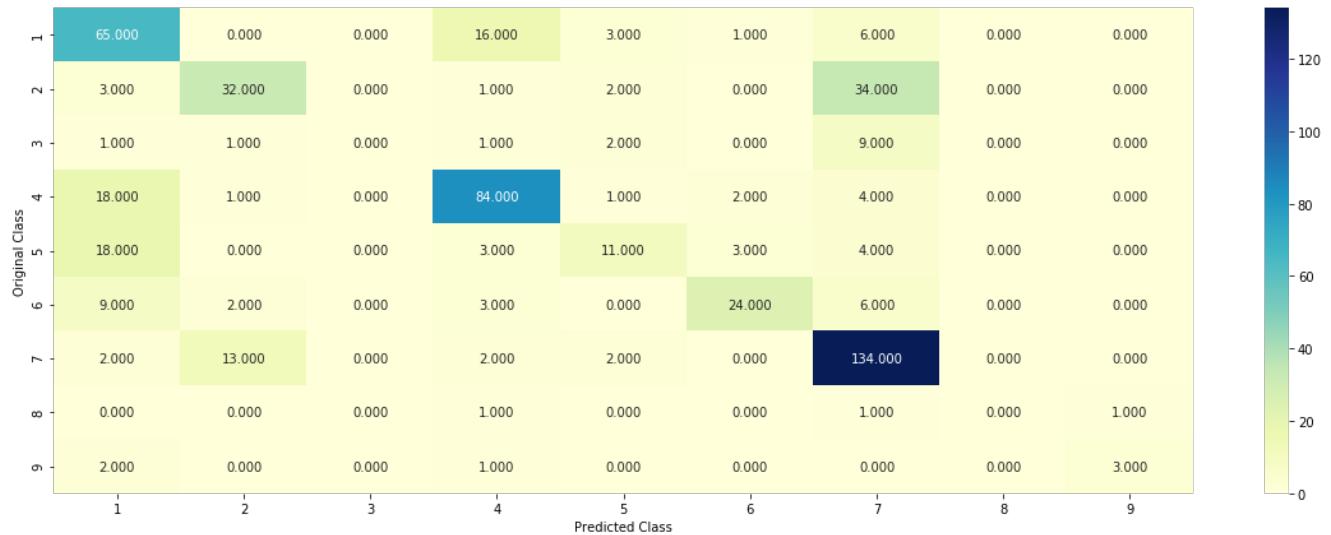
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

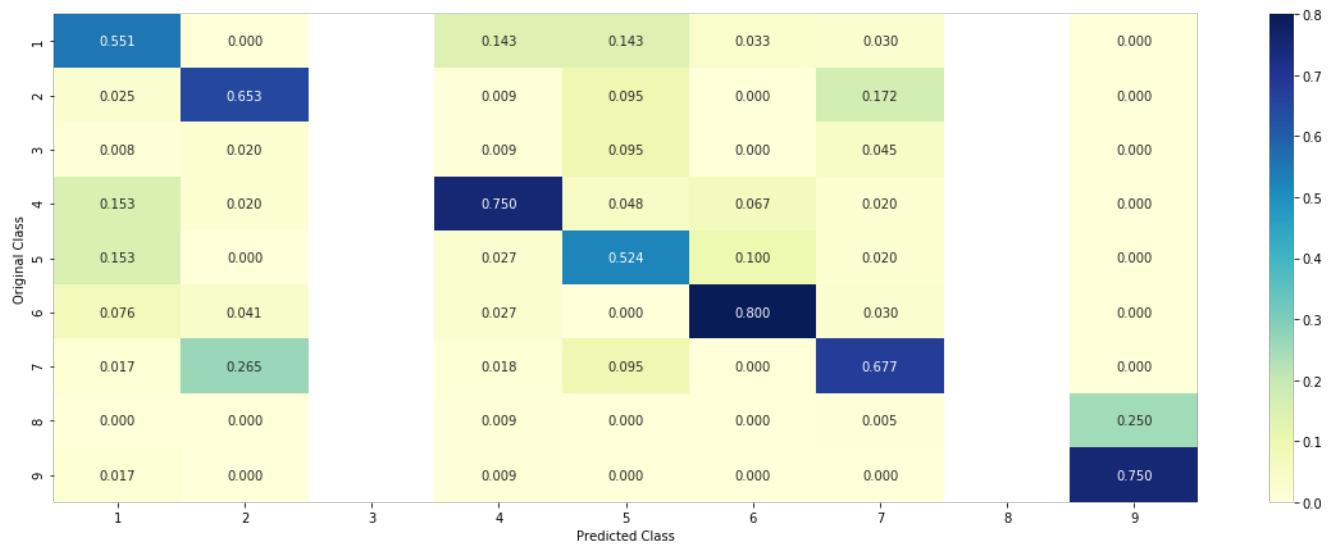
```

Log loss : 1.019533719050228
Number of mis-classified points : 0.33646616541353386
----- Confusion matrix -----

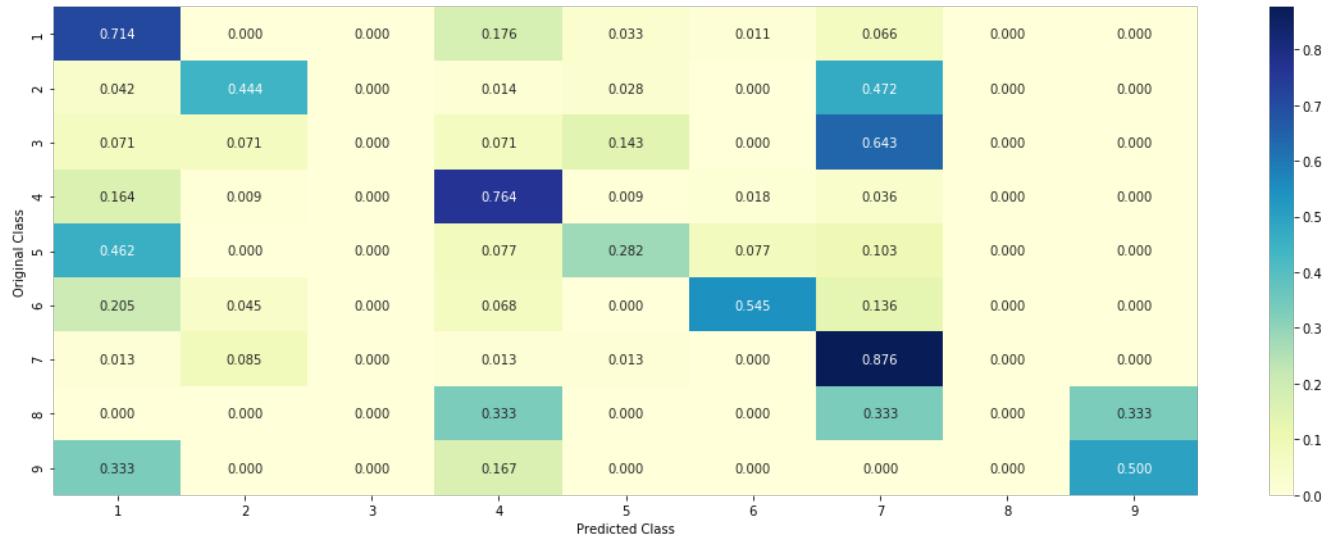
```



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



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



#### 4.3.2.3. Feature Importance, Correctly Classified point

In [0]:

```

test_point_index = 28
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(np.abs(-clf.coef_))[predicted_cls-1][:, :no_feature]
print("-" * 50)
get_imfeature_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 : 4
Predicted Class Probabilities: [[0.2437 0.0175 0.0105 0.5787 0.0261 0.0152 0.097 0.0064 0.0049]]
Actual Class : 4
-----
Out of the top 500 features 0 are present in query point

```

#### 4.3.2.4. Feature Importance, Incorrectly Classified point

In [0]:

```

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 43
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(np.abs(-clf.coef_))[predicted_cls-1][:, :no_feature]
print("-" * 50)
get_imfeature_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: [[7.148e-01 2.940e-02 1.260e-02 2.130e-01 6.500e-03 7.000e-03 7.700e-03
8.400e-03 6.000e-04]]
Actual Class : 4
-----

```

35 Text feature [001] present in test data point [True]  
Out of the top 500 features 1 are present in query point

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper parameter tuning

In [0]:

```

# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.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/
# -----

```

```

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

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

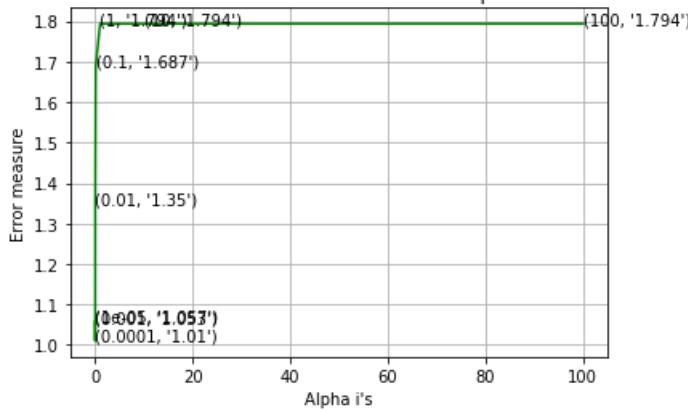
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', 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 C = 1e-05
Log Loss : 1.0573262254626619
for C = 0.0001
Log Loss : 1.0099782136421305
for C = 0.001
Log Loss : 1.052814744876514
for C = 0.01
Log Loss : 1.349792060738385
for C = 0.1
Log Loss : 1.687442354117265
for C = 1
Log Loss : 1.7935690448506416
for C = 10
Log Loss : 1.793569207141754
for C = 100
Log Loss : 1.7935658748165901

```



For values of best alpha = 0.0001 The train log loss is: 0.31127548717619086  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.0099782136421305  
 For values of best alpha = 0.0001 The test log loss is: 1.0666569223612394

#### 4.4.2. Testing model with best hyper parameters

In [0]:

```
# read more about support vector machines with linear kernels 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)

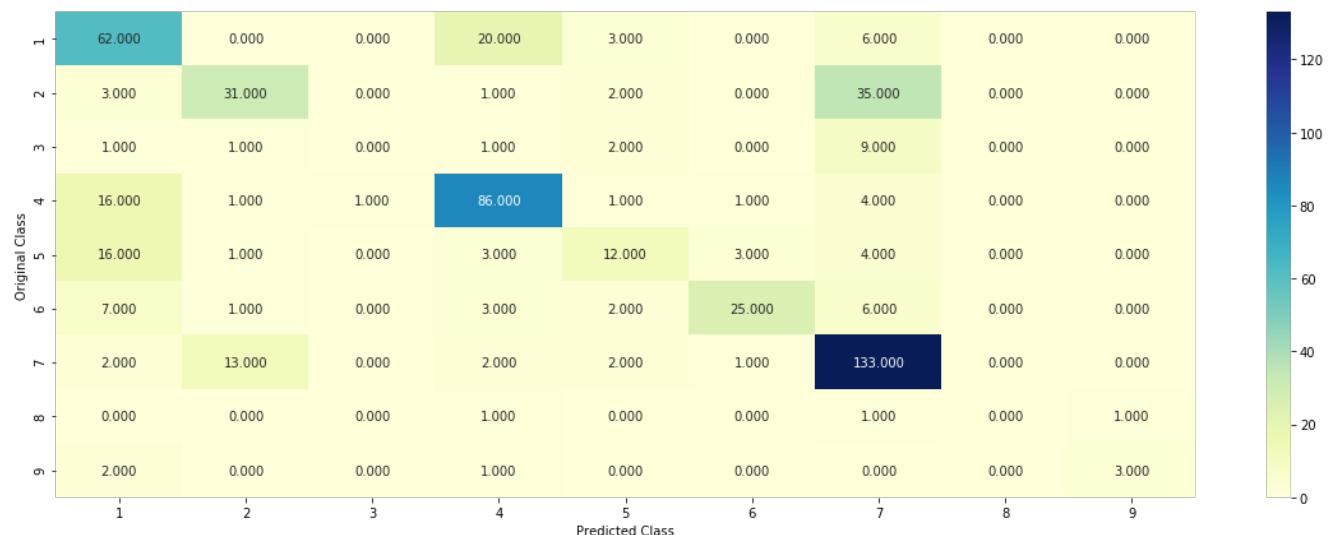
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----


# 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)
```

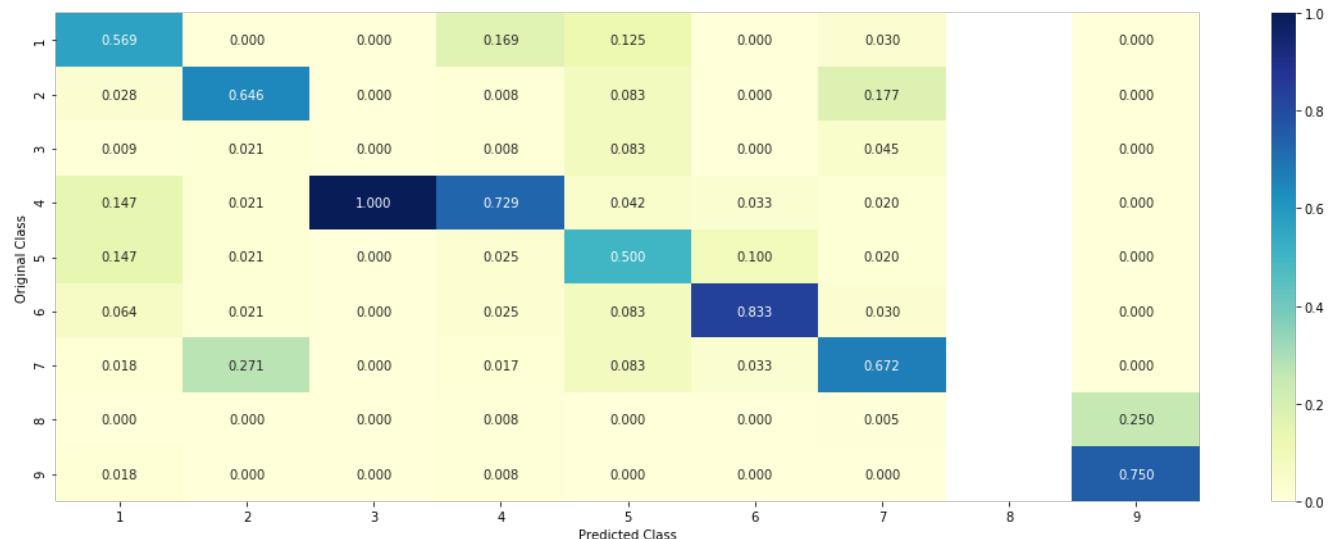
Log loss : 1.0099782136421305

Number of mis-classified points : 0.3383458646616541

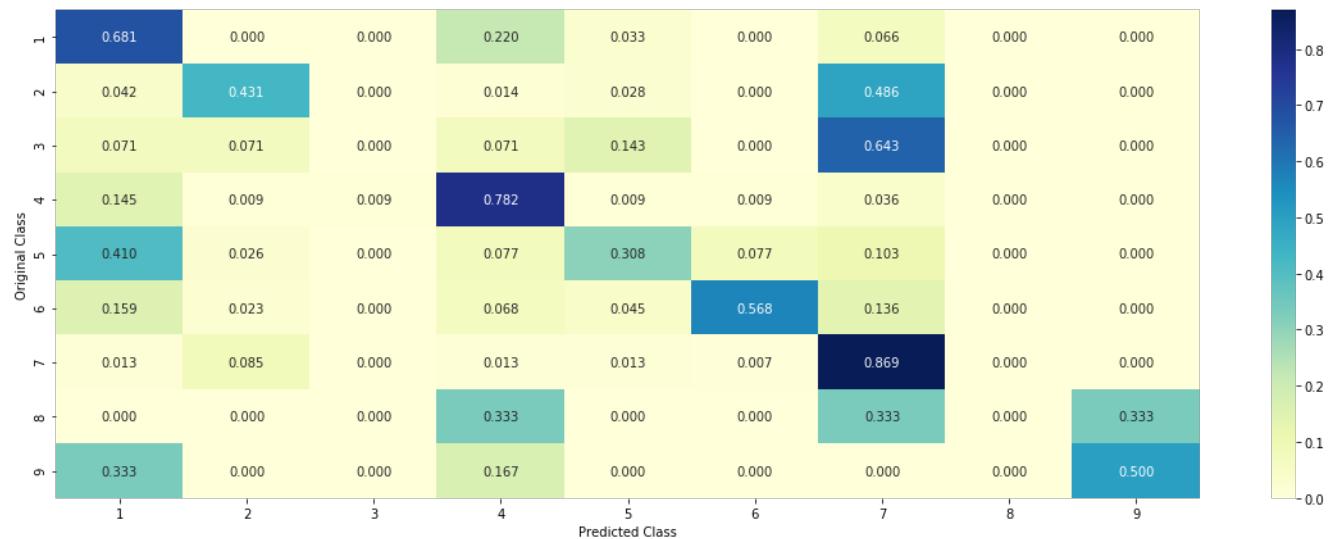
----- Confusion matrix -----



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



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



### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

In [0]:

```
test_point_index = 89
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(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-" * 50)
get_imptfeature_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 : 6

Predicted Class Probabilities: [[0.0394 0.0289 0.0057 0.0214 0.0274 0.8584 0.0114 0.0059 0.0015]]

Actual Class : 6

Out of the top 500 features 0 are present in query point

#### 4.3.3.2. For Incorrectly classified point

In [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 142
# 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(np.abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-" * 50)
get_imptfeature_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: [[0.5933 0.0428 0.0087 0.1334 0.0099 0.1436 0.0512 0.0115 0.0056]]
Actual Class : 6
-----
Out of the top 500 features 0 are present in query point
```

## 4.5 Random Forest Classifier

### 4.5.1. Hyper parameter tuning (With One hot Encoding)

In [0]:

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

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

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

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


# 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 : 1.1844176568956086
for n_estimators = 100 and max depth =  10
Log Loss : 1.2067824578402169
for n_estimators = 200 and max depth =  5
Log Loss : 1.1737772412868388
for n_estimators = 200 and max depth =  10
Log Loss : 1.2003330145202287
for n_estimators = 500 and max depth =  5
Log Loss : 1.1710518301956077
for n_estimators = 500 and max depth =  10
Log Loss : 1.1952815497302567
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1687302732054692
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1962263064448122
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1681080860569861
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1964386832932075
For values of best estimator =  2000 The train log loss is: 0.8294764907363178
For values of best estimator =  2000 The cross validation log loss is: 1.1681080860569861
For values of best estimator =  2000 The test log loss is: 1.2015789183370509

```

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

In [0]:

```
# -----
# default parameters
```

```

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

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

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

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

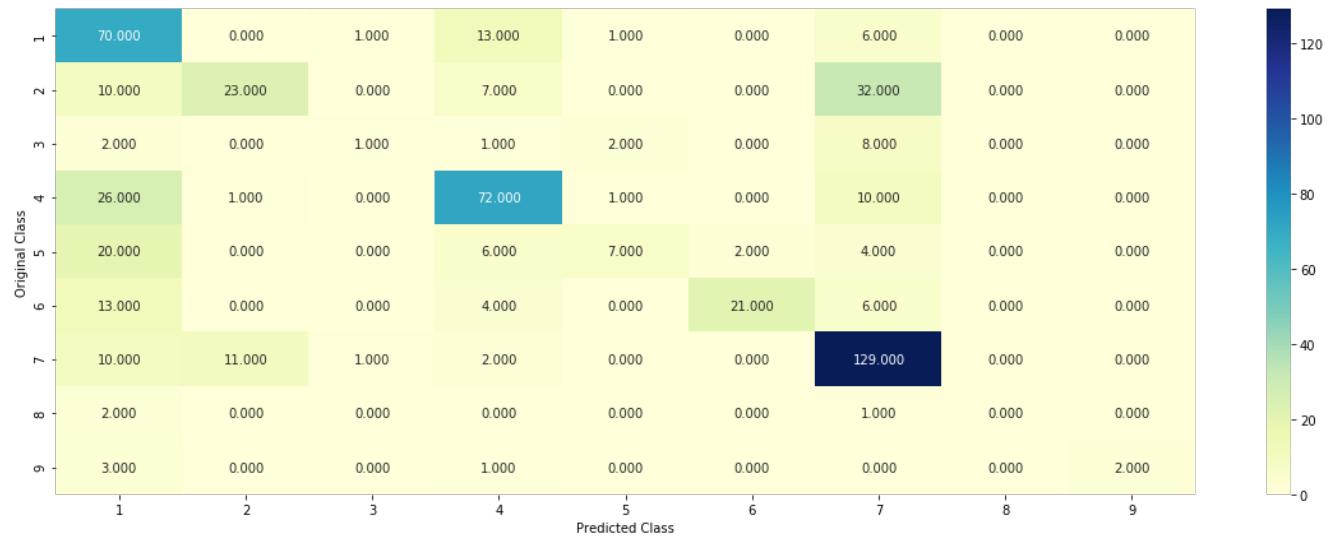

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

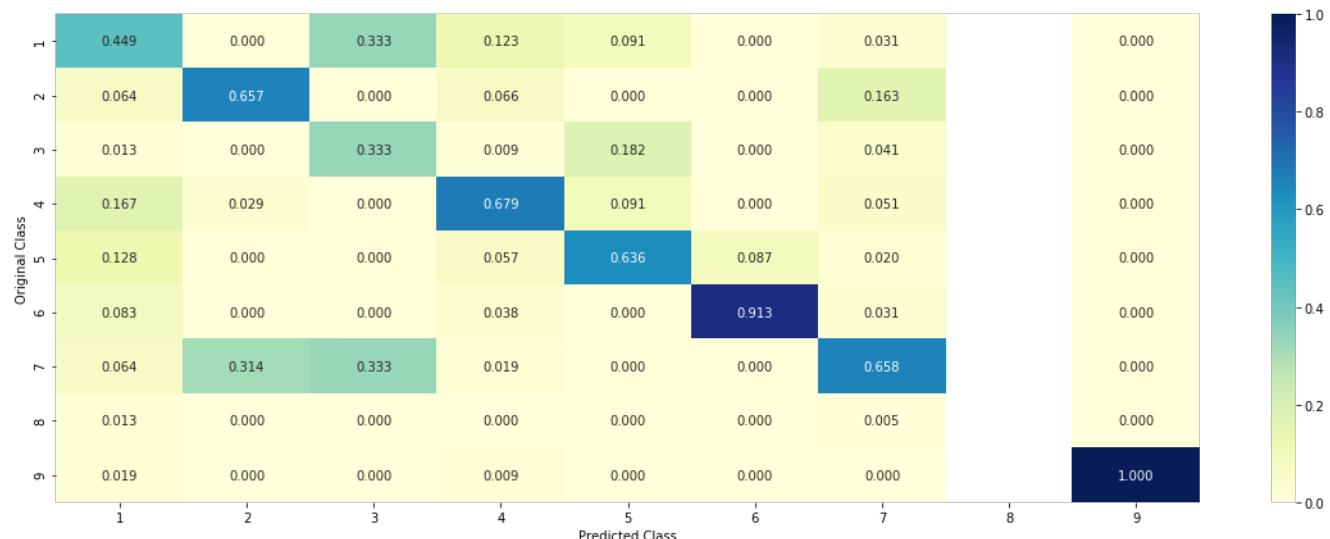
Log loss : 1.1681080860569861

Number of mis-classified points : 0.3890977443609023

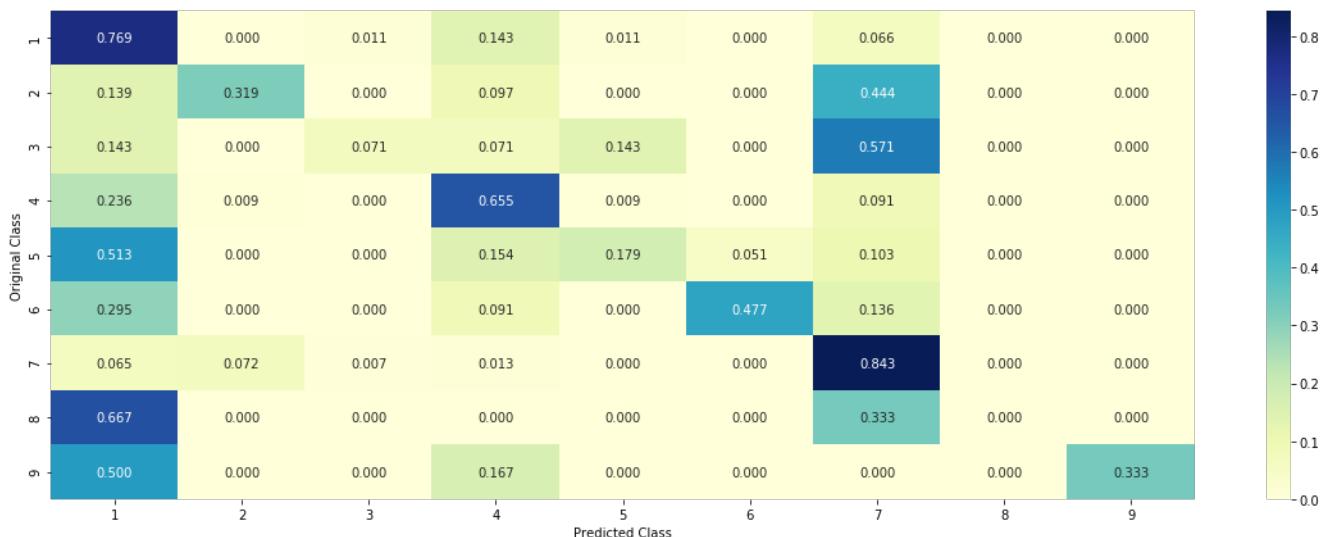
----- Confusion matrix -----



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



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



### 4.5.3. Feature Importance

#### 4.5.3.1. Correctly Classified point

In [0]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-" * 50)
get_imptfeature_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 : 4
Predicted Class Probabilities: [[0.1522 0.0422 0.0115 0.591  0.0448 0.0443 0.0661 0.0075 0.0403]]
Actual Class : 4
-----
Out of the top 100 features 0 are present in query point
```

#### 4.5.3.2. Incorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-" * 50)
get_imptfeature_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: [[0.3206 0.0663 0.0144 0.2001 0.0793 0.2768 0.03 0.0067 0.0059]]
Actual Class : 4
-----
5 Text feature [069] present in test data point [True]
20 Text feature [051] present in test data point [True]
70 Text feature [05] present in test data point [True]
91 Text feature [03] present in test data point [True]
Out of the top 100 features 4 are present in query point

```

#### 4.5.3. Hyper parameter tuning (With Response Coding)

In [0]:

```

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

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

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

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


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


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

```

```

plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```

```

for n_estimators = 10 and max depth =  2
Log Loss : 2.0158554172562813
for n_estimators = 10 and max depth =  3
Log Loss : 1.590211675472542
for n_estimators = 10 and max depth =  5
Log Loss : 1.186936591976196
for n_estimators = 10 and max depth =  10
Log Loss : 1.7545791123097665
for n_estimators = 50 and max depth =  2
Log Loss : 1.5914579287729067
for n_estimators = 50 and max depth =  3
Log Loss : 1.3150125593215625
for n_estimators = 50 and max depth =  5
Log Loss : 1.219107685205031
for n_estimators = 50 and max depth =  10
Log Loss : 1.6195884702243015
for n_estimators = 100 and max depth =  2
Log Loss : 1.4660223892653226
for n_estimators = 100 and max depth =  3
Log Loss : 1.352600293328983
for n_estimators = 100 and max depth =  5
Log Loss : 1.2249395520084436
for n_estimators = 100 and max depth =  10
Log Loss : 1.6594877205728977
for n_estimators = 200 and max depth =  2
Log Loss : 1.508459982832084
for n_estimators = 200 and max depth =  3
Log Loss : 1.3904037498161141
for n_estimators = 200 and max depth =  5
Log Loss : 1.2863400706283818
for n_estimators = 200 and max depth =  10
Log Loss : 1.693627214974659
for n_estimators = 500 and max depth =  2
Log Loss : 1.5549497485993444
for n_estimators = 500 and max depth =  3
Log Loss : 1.4545016609230375
for n_estimators = 500 and max depth =  5
Log Loss : 1.3133552372897308
for n_estimators = 500 and max depth =  10
Log Loss : 1.7227087400353536
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5547563883131421
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4744914003771077
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3225232989793625
for n_estimators = 1000 and max depth =  10
Log Loss : 1.742149353356783
For values of best alpha =  10 The train log loss is: 0.08649921173521453
For values of best alpha =  10 The cross validation log loss is: 1.186936591976196
For values of best alpha =  10 The test log loss is: 1.2433838325862103

```

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

In [0]:

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

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

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

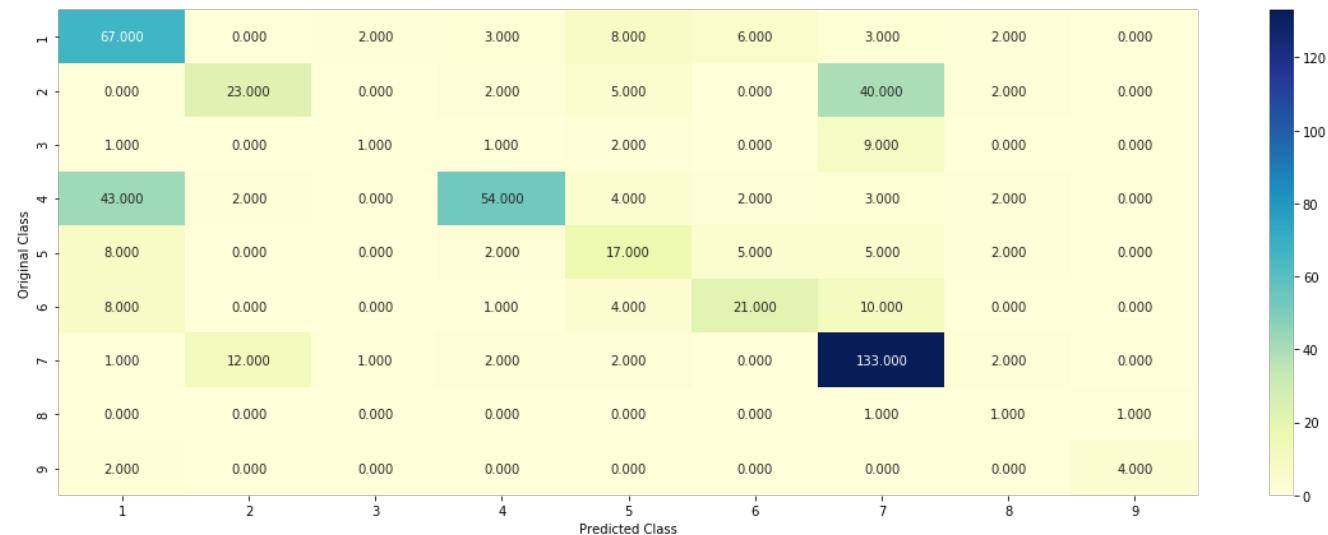
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----
```

clf = RandomForestClassifier(max\_depth=max\_depth[int(best\_alpha%4)], n\_estimators=alpha[int(best\_alpha/4)], criterion='gini', max\_features='auto', random\_state=42)  
predict\_and\_plot\_confusion\_matrix(train\_x\_responseCoding, train\_y, cv\_x\_responseCoding, cv\_y, clf)

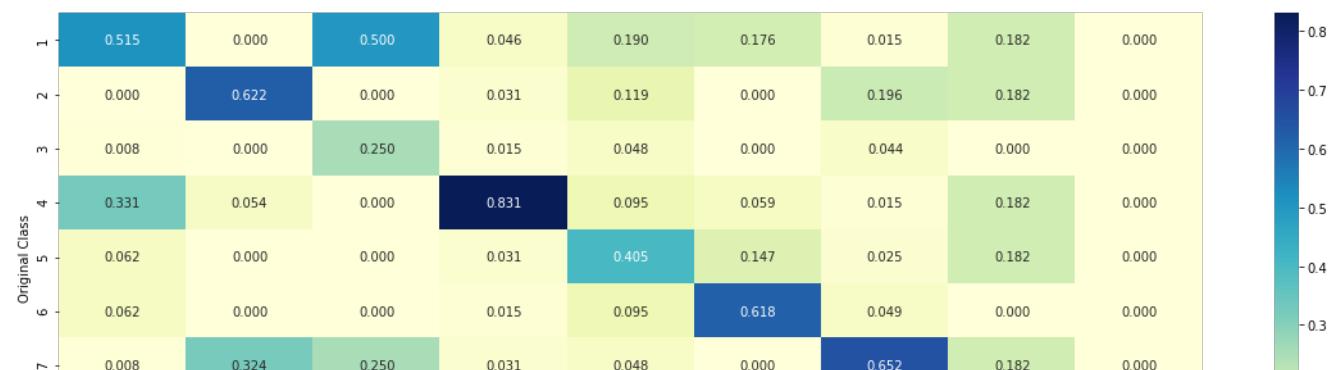
Log loss : 1.186936591976196

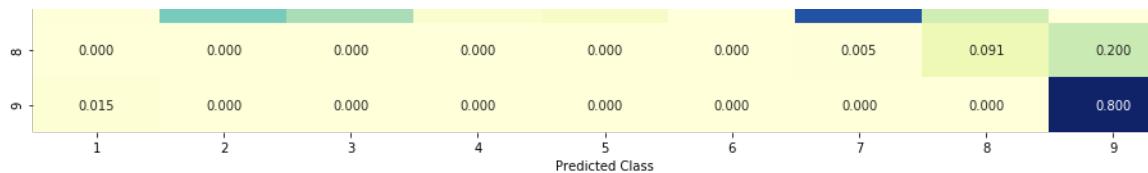
Number of mis-classified points : 0.3966165413533835

----- Confusion matrix -----

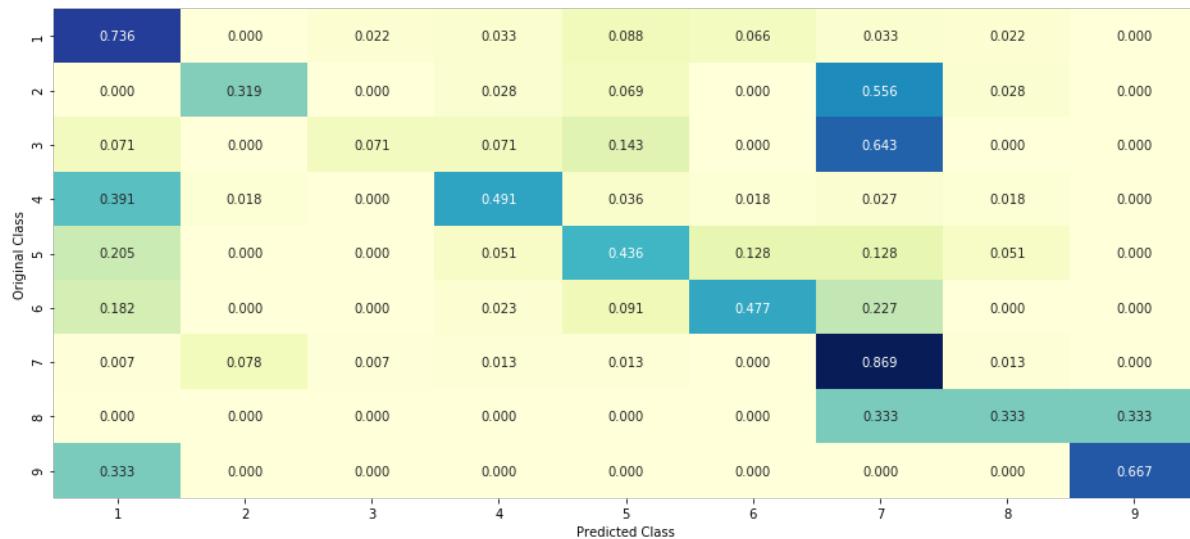


----- Precision matrix (Column Sum=1) -----





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



#### 4.5.5. Feature Importance

##### 4.5.5.1. Correctly Classified point

In [0]:

```

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

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

```

Predicted Class : 4  
Predicted Class Probabilities: [[0.1684 0.0211 0.0667 0.5484 0.0092 0.0336 0.0042 0.1033 0.0452]]  
Actual Class : 4

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

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

#### 4.5.5.2. Incorrectly Classified point

In [0]:

```
test_point_index = 52
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: [[0.0351 0.083  0.1092 0.0259 0.0522 0.0502 0.5816 0.0344 0.0283]]
Actual Class : 2
-----
```

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

## 4.7 Stack the models

### 4.7.1 testing with hyper parameter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, 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-intuiti
on-1/
#-----

# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modul
es/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-deri
vation-copy-8/
# -----


# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modul
es/generated/sklearn.ensemble.RandomForestClassifier.html
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impu
rity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_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)
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_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.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: 1.03
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.16
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 1.709
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.297
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.188
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.525
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.913

```

#### 4.7.2 testing the model with the best hyper parameters

In [0]:

```

lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=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))

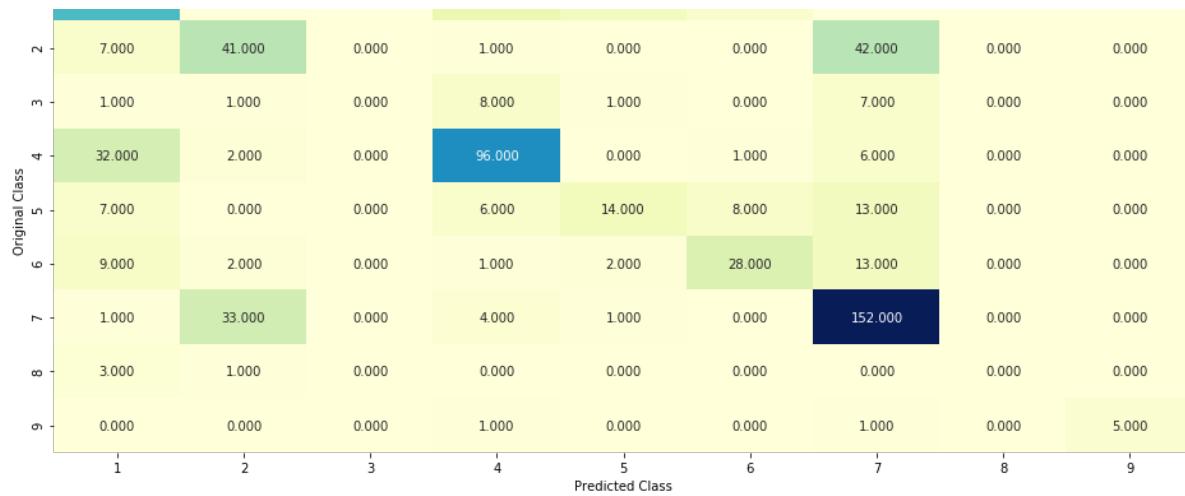
```

```

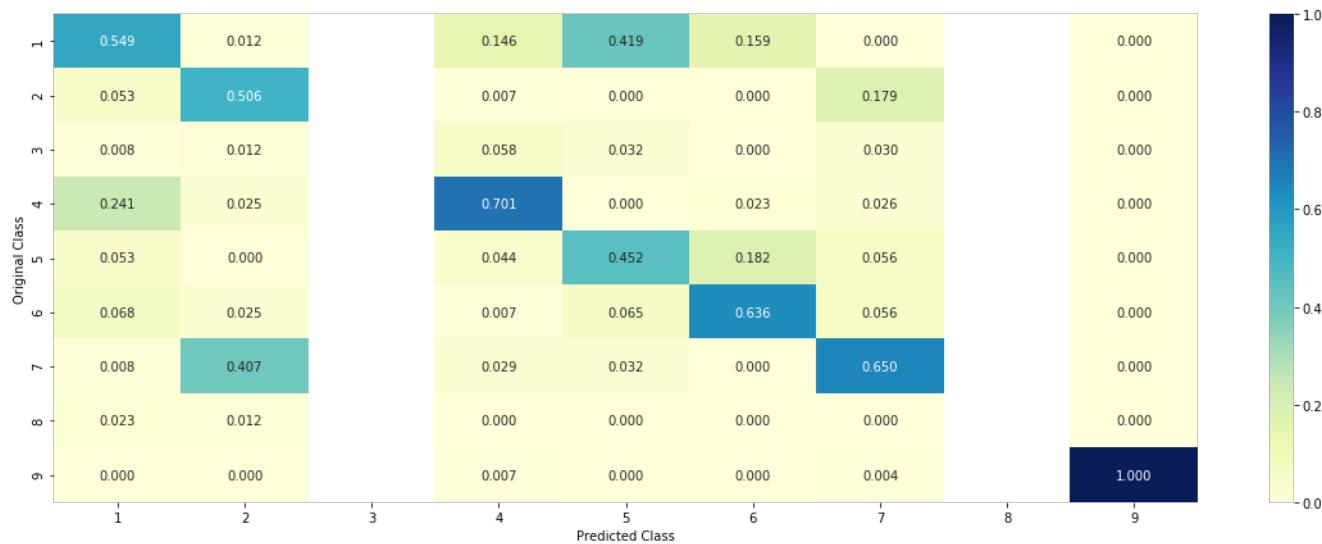
Log loss (train) on the stacking classifier : 0.333407851644082
Log loss (CV) on the stacking classifier : 1.1877922711134676
Log loss (test) on the stacking classifier : 1.2813434488251318
Number of missclassified point : 0.3849624060150376
----- Confusion matrix -----

```

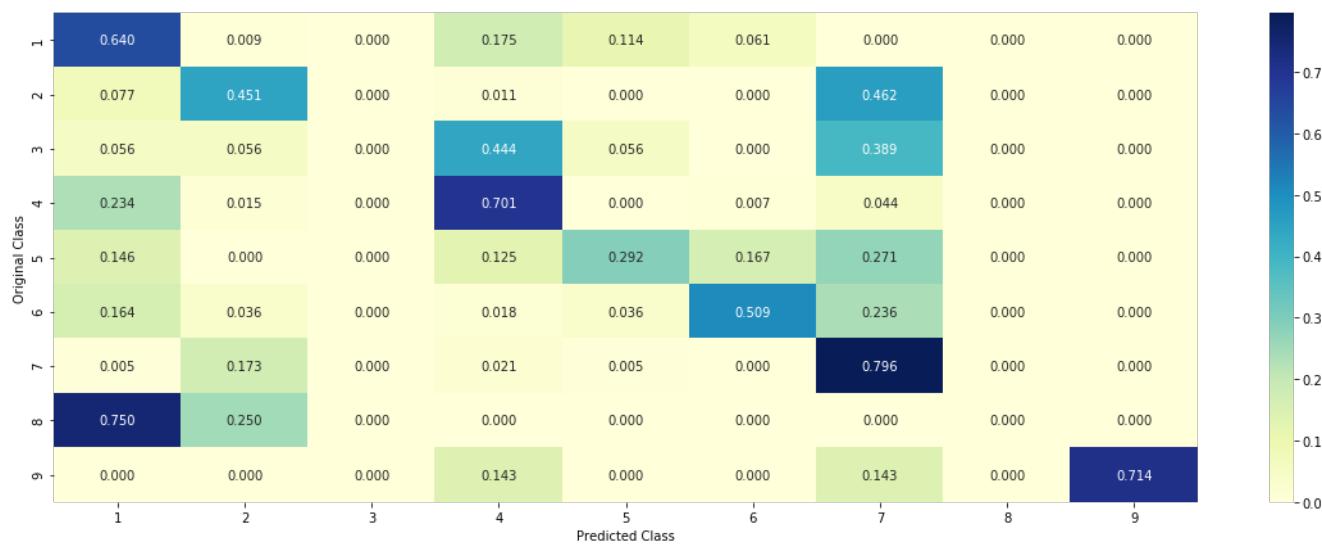
73.000	1.000	0.000	20.000	13.000	7.000	0.000	0.000
--------	-------	-------	--------	--------	-------	-------	-------



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



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



#### 4.7.3 Maximum Voting classifier

In [0]:

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

```
t')
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))
```

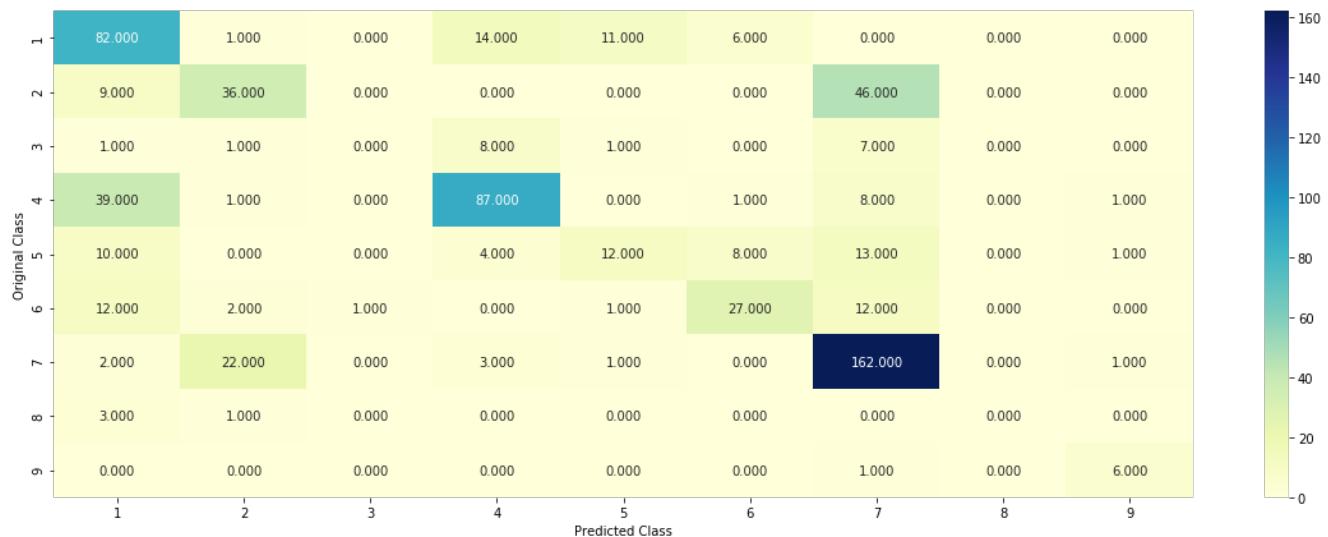
Log loss (train) on the VotingClassifier : 0.7948340118488088

Log loss (CV) on the VotingClassifier : 1.1777574764847945

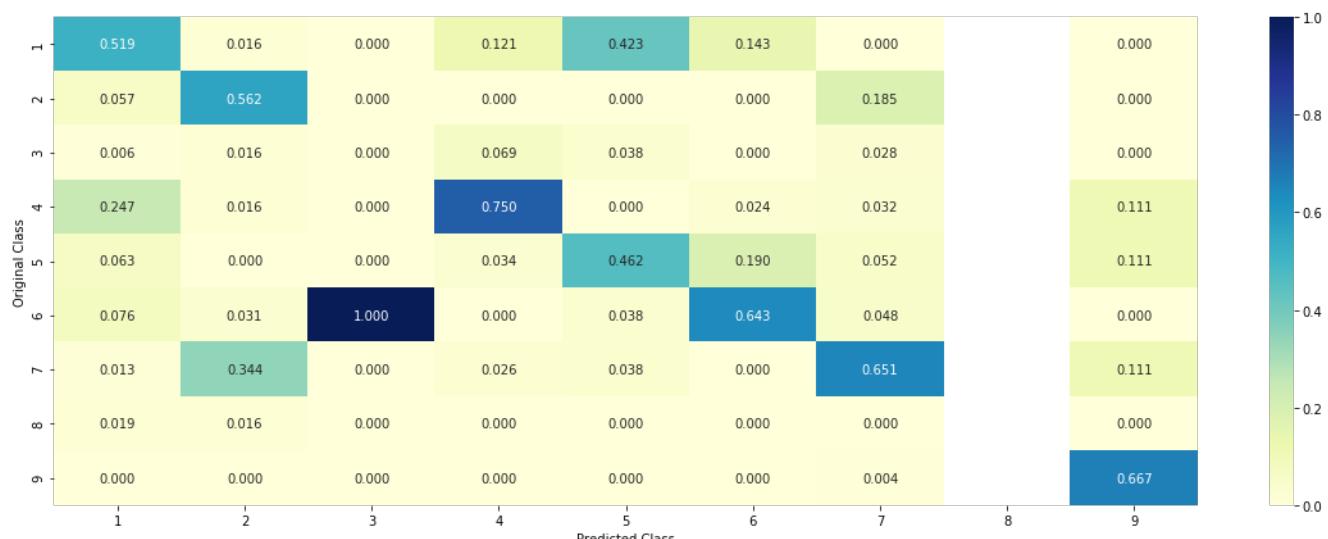
Log loss (test) on the VotingClassifier : 1.2177900042055676

Number of missclassified point : 0.3804511278195489

----- Confusion matrix -----



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



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

