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

1.3. Real-world/Business objectives and constraints.

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

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=ur
n%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&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&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
path = "/content/drive/My Drive/Colab Notebooks/AppliedAI/Personalized_Cancer_Diagnosis/"
```

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
# from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV,StratifiedKFold
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/sklearn/externals/six.py:31: DeprecationWarning: 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/).", DeprecationWarning)
```

In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
table = PrettyTable(field_names=['Vectorizer','Model','Test Log Loss',
                                 'Improvement'])
```

In [7]:

```
print(table)
```

```
+------------+-------+---------------+-------------+
| Vectorizer | Model | Test Log Loss | Improvement |
+------------+-------+---------------+-------------+
+------------+-------+---------------+-------------+
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [0]:

```
data = pd.read_csv(path+'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 seprator in this file
data_text =pd.read_csv(path+"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.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.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 : 28.80702000000001 seconds
```

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 lineag...

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	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

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

In [0]:

```
print(result.shape)
print(train_df.shape)
print(test_df.shape)
print(cv_df.shape)
```

```
(3321, 5)
(2124, 5)
(665, 5)
(532, 5)
```

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
```

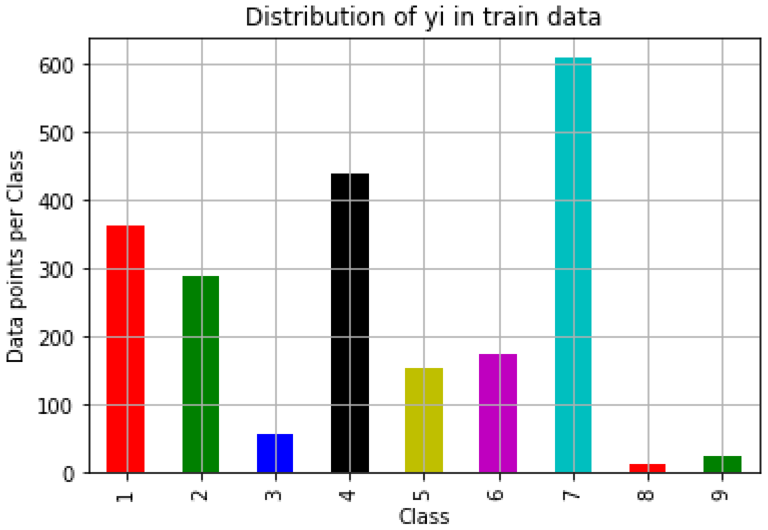
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [0]: train_df['Class'].head(8)
```

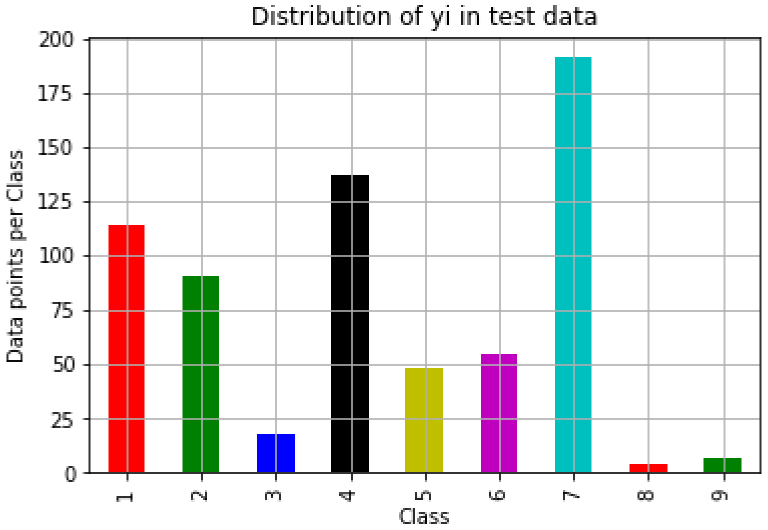
```
Out[0]: 1645    7
        2747    2
        2222    4
        943     1
        638     4
        2424    1
        838     2
        2960    2
        Name: Class, dtype: int64
```

In [0]:

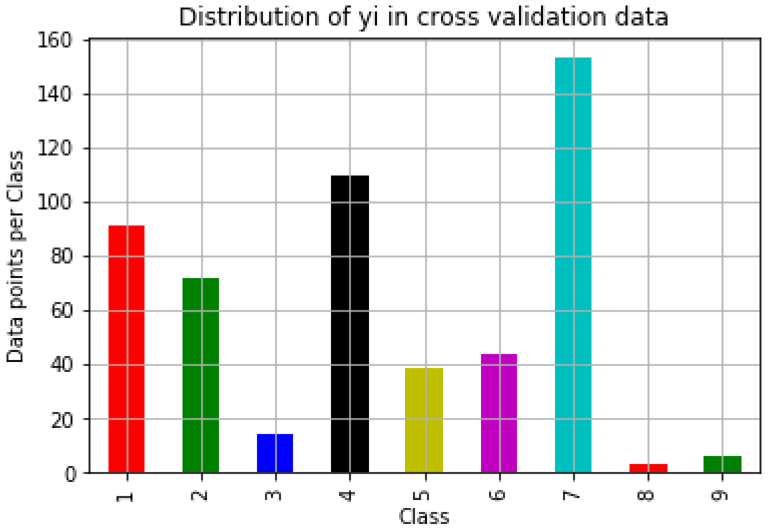
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar',color=list(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=list(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=list(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 probabilites randomly such that they sum to 1.

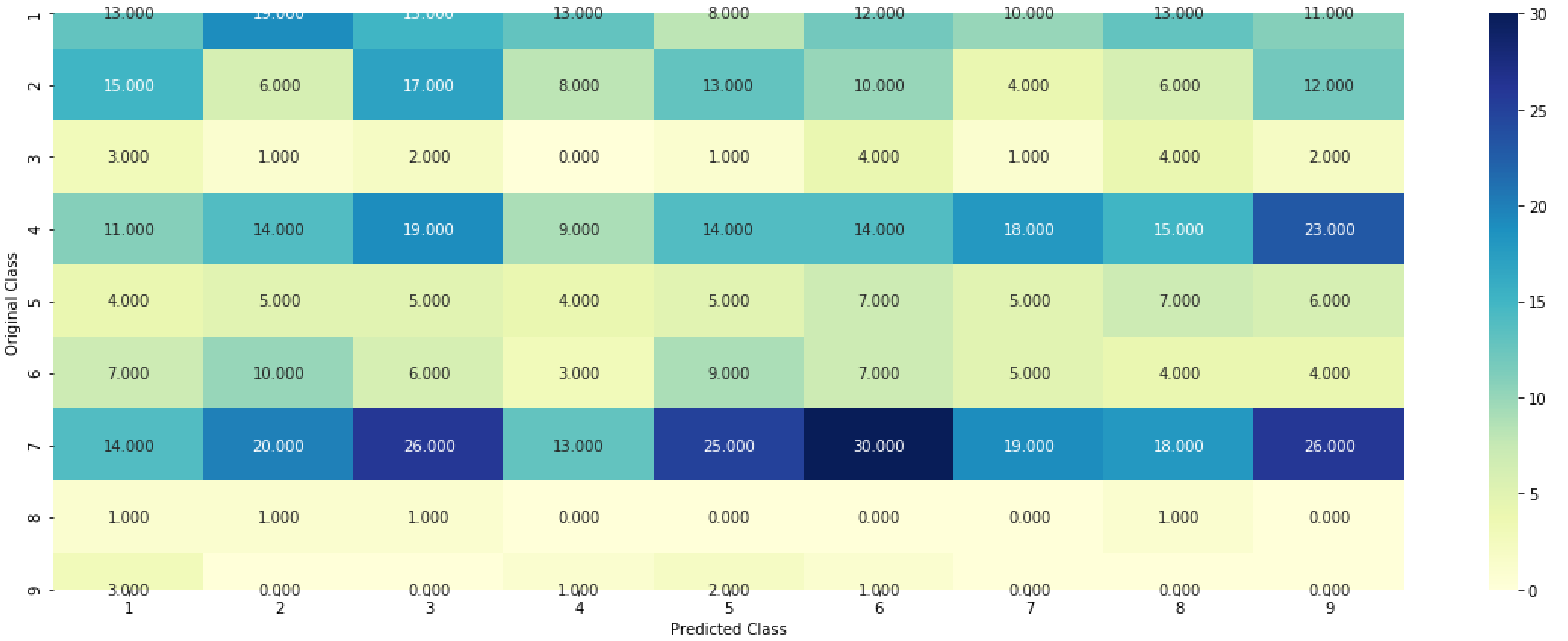
In [0]:

```
# 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)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i
    # are predicted class j

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

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

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

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

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

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

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

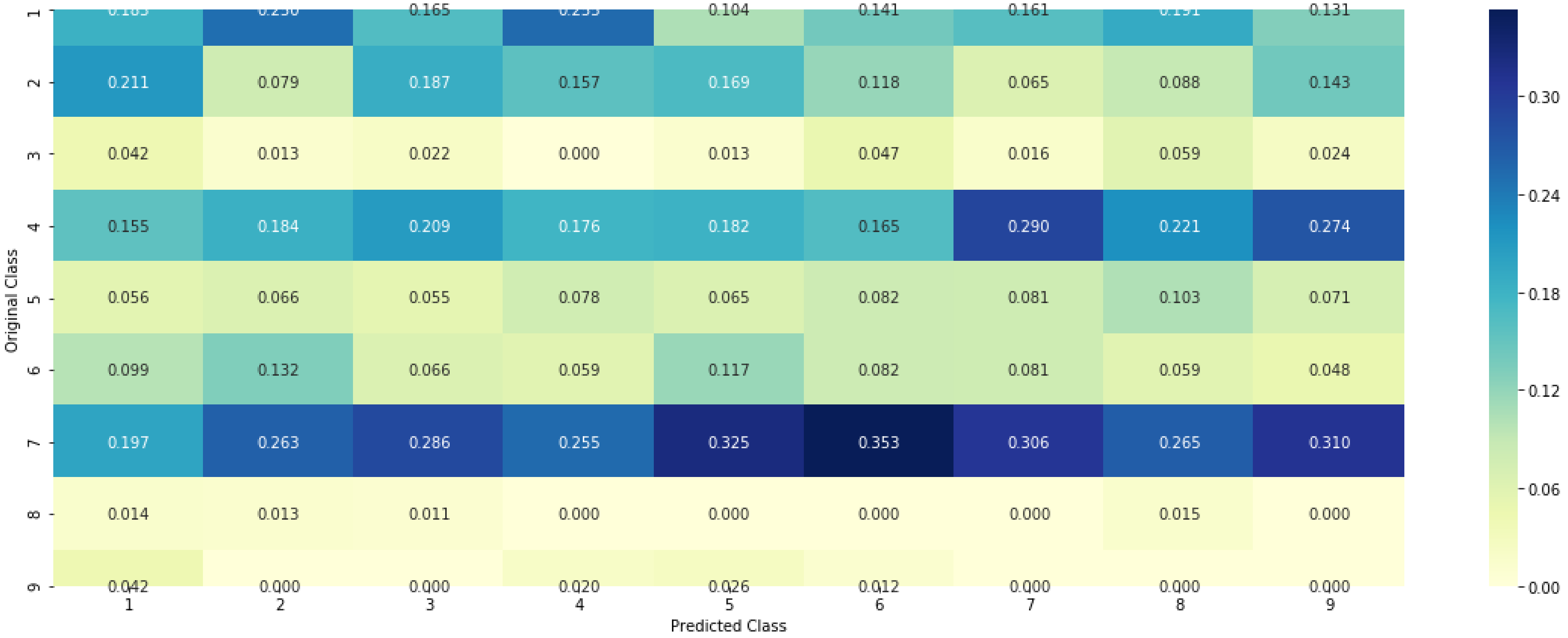
In [0]:

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

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


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

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

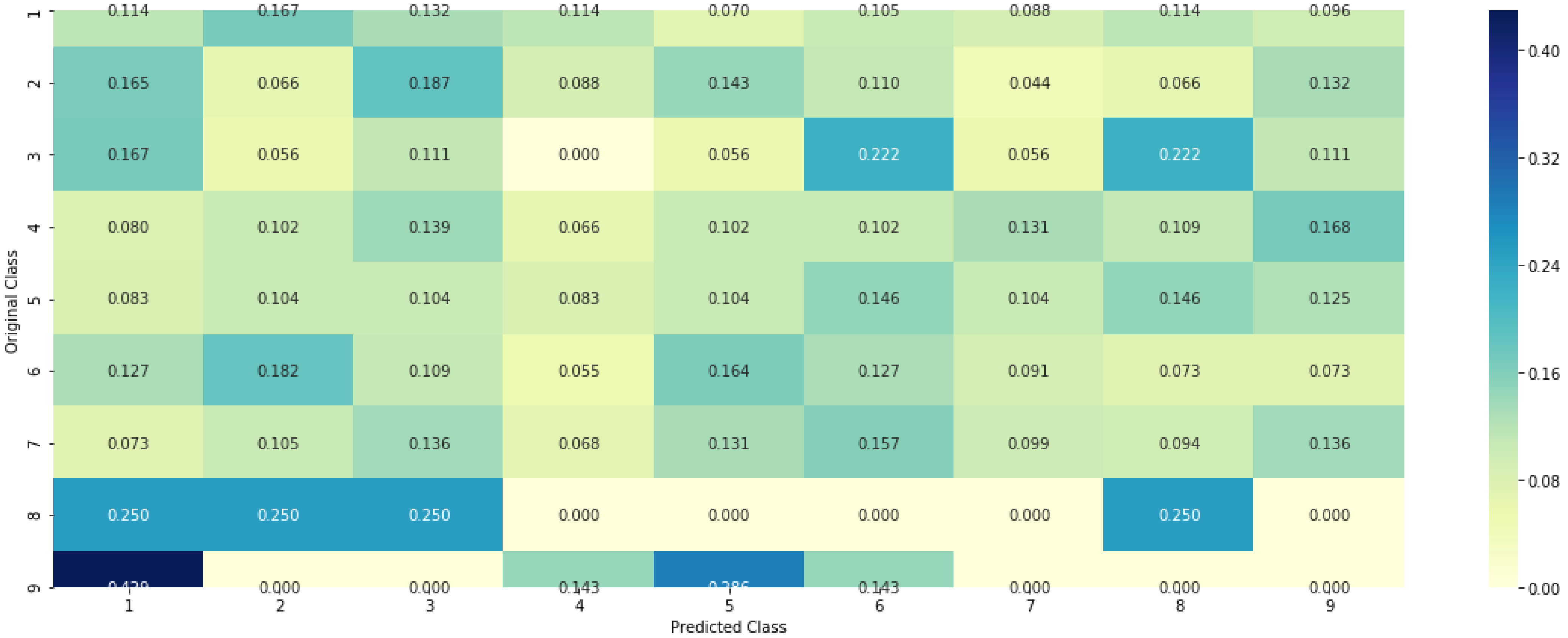
```
Log loss on Cross Validation Data using Random Model 2.471380780366412
Log loss on Test Data using Random Model 2.4897763497807386
-------------------- Confusion matrix --------------------
```



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



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



In [8]:

```
table.add_row(['NAN','Random',2.48,0])
print(table)
```

```
+------------+--------+---------------+-------------+
| Vectorizer | Model  | Test Log Loss | Improvement |
+------------+--------+---------------+-------------+
|    NAN     | Random |      2.48     |      0      |
+------------+--------+---------------+-------------+
```

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 occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / 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 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.0378787878787
8788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.05102040816326530
7, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181
816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060606060606060
8, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.0691823899371069
17, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.2715231788079470
2, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.2999999999999999
9, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [0]:

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

```
Number of Unique Genes : 232
BRCA1     176
TP53       98
EGFR       96
PTEN       77
BRCA2      76
KIT        64
BRAF       63
ERBB2      48
ALK        41
PDGFRA     38
Name: Gene, dtype: int64
```

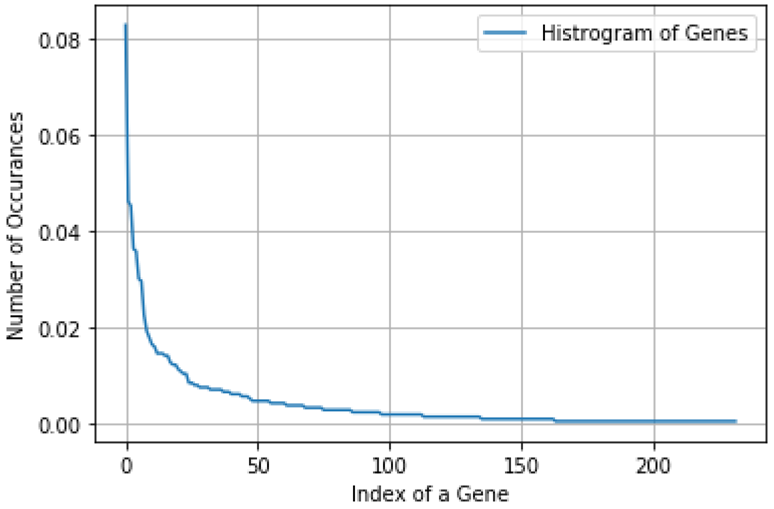
In [0]:

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

```
Ans: There are 232 different categories of genes in the train data, and they are distibuted as follows
```

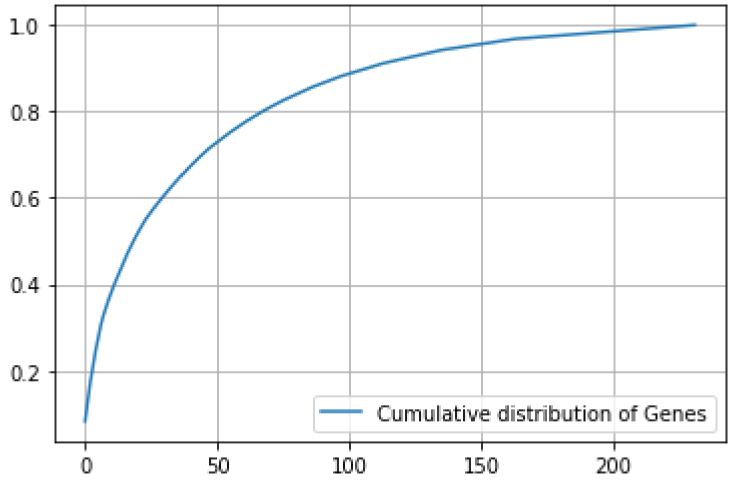
In [0]:

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



In [0]:

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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [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 respone coding method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [0]:

```
# one-hot encoding of Gene feature.
# gene_vectorizer = CountVectorizer()
gene_vectorizer = TfidfVectorizer(max_features=1000)
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]:

```
961      KDM5C
297      CHEK2
1209    PIK3CA
957      KDM5C
257       EGFR
Name: Gene, dtype: object
```

In [0]:

```
gene_vectorizer.get_feature_names()
```

```
Out[0]: ['abl1',
         'acvr1',
         'ago2',
         'akt1',
         'akt2',
         'akt3',
         'alk',
         'apc',
         'ar',
         'araf',
         'arid1b',
         'arid2',
         'arid5b',
         'asxl1',
         'asxl2',
         'atm',
         'atr',
         'atrx',
         'aurka',
         'aurkb',
         'axin1',
         'axl',
         'b2m',
         'bap1',
         'bcor',
         'braf',
         'brca1',
         'brca2',
         'brd4',
         'brip1',
         'btk',
         'card11',
         'casp8',
         'cbl',
         'ccnd1',
         'ccnd2',
         'ccnd3',
         'cdh1',
         'cdk12',
         'cdk4',
         'cdk6',
         'cdk8',
         'cdkn1a',
         'cdkn1b',
         'cdkn2a',
         'cdkn2b',
         'cdkn2c',
         'cebpa',
         'chek2',
         'cic',
         'crebbp',
         'ctcf',
         'ctnnb1',
         'ddr2',
         'dicer1',
         'dnmt3a',
         'dnmt3b',
         'dusp4',
         'egfr',
         'elf3',
         'ep300',
         'epas1',
         'epcam',
         'erbb2',
         'erbb3',
         'erbb4',
         'ercc2',
         'ercc3',
         'ercc4',
         'erg',
         'errfi1',
         'esr1',
         'etv1',
         'etv6',
         'ewsr1',
         'ezh2',
         'fanca',
         'fancc',
         'fat1',
         'fbxw7',
         'fgf19',
         'fgf3',
         'fgf4',
         'fgfr1',
         'fgfr2',
         'fgfr3',
         'fgfr4',
         'flt1',
         'flt3',
         'foxa1',
         'foxp1',
         'fubp1',
         'gata3',
         'gnaq',
         'gnas',
         'h3f3a',
         'hist1h1c',
         'hla',
         'hnf1a',
         'hras',
         'idh1',
         'idh2',
         'igf1r',
         'ikzf1',
         'il7r',
         'jak1',
         'jak2',
         'kdm5a',
         'kdm5c',
         'kdm6a',
         'kdr',
         'keap1',
         'kit',
         'kmt2a',
         'kmt2b',
         'kmt2c',
         'kmt2d',
         'knstrn',
         'kras',
         'lats1',
         'lats2',
         'map2k1',
         'map2k2',
         'map2k4',
         'map3k1',
         'mapk1',
         'mdm2',
         'mdm4',
         'med12',
         'mef2b',
         'men1',
         'met',
         'mlh1',
         'mpl',
         'msh2',
         'msh6',
         'mtor',
         'myc',
         'mycn',
         'myd88',
         'myod1',
         'ncor1',
         'nf1',
         'nf2',
         'nfe2l2',
         'nfkbia',
         'nkx2',
         'notch1',
         'notch2',
         'npm1',
         'nras',
         'nsd1',
         'ntrk1',
         'ntrk3',
         'nup93',
         'pak1',
         'pbrm1',
         'pdgfra',
         'pdgfrb',
         'pik3ca',
         'pik3cb',
         'pik3cd',
         'pik3r1',
         'pik3r2',
         'pik3r3',
         'pim1',
         'pms2',
         'pole',
         'ppm1d',
         'ppp2r1a',
         'ppp6c',
         'prdm1',
         'ptch1',
         'pten',
         'ptpn11',
         'ptprd',
         'ptprt',
         'rac1',
         'rad21',
         'rad50',
         'rad51b',
         'rad51c',
         'rad51d',
         'rad54l',
         'raf1',
         'rara',
         'rasa1',
         'rb1',
         'ret',
         'rheb',
         'rhoa',
         'rit1',
         'rnf43',
         'ros1',
         'runx1',
         'rxra',
         'rybp',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stag2',
'stat3',
'stk11',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'whsc1',
'xpo1',
'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, 231)
```

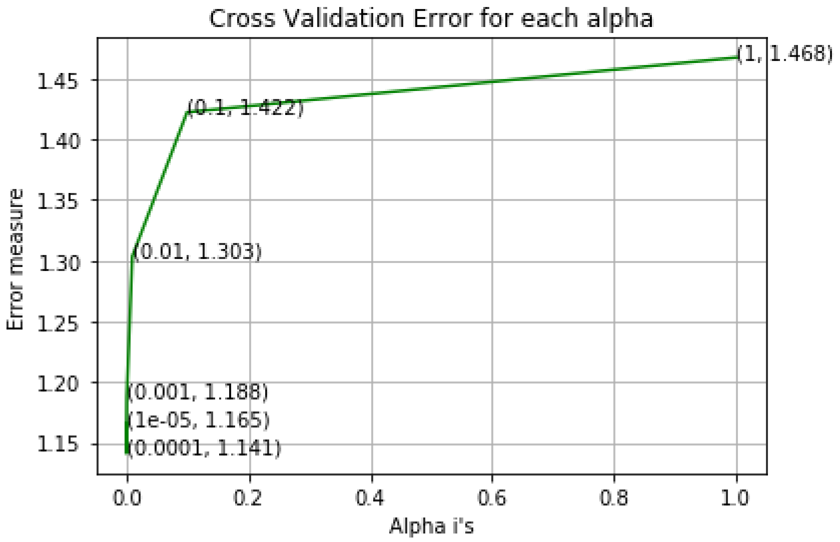
Q4. How good is this gene feature in predicting y_i?

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))

# Improvement in test log = 0.01
```

```
For values of alpha =  1e-05 The log loss is: 1.1651790502948691
For values of alpha =  0.0001 The log loss is: 1.141113608467628
For values of alpha =  0.001 The log loss is: 1.187816559069162
For values of alpha =  0.01 The log loss is: 1.3031596813546533
For values of alpha =  0.1 The log loss is: 1.4224359370863766
For values of alpha =  1 The log loss is: 1.467619854974226
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0174149002829747
For values of best alpha =  0.0001 The cross validation log loss is: 1.141113608467628
For values of best alpha =  0.0001 The test log loss is: 1.1932450996185502
```

In [0]:
```
# Improvement of 0.01 on test log loss
```

In [9]:
```
table.add_row(['Gene Feature One-Hot','Linear SVM', 1.19, 0.01])
print(table)
```

```
+----------------------+------------+---------------+-------------+
|      Vectorizer      |   Model    | Test Log Loss | Improvement |
+----------------------+------------+---------------+-------------+
|         NAN          |   Random   |      2.48     |      0      |
| Gene Feature One-Hot | Linear SVM |      1.19     |     0.01    |
+----------------------+------------+---------------+-------------+
```

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  232  genes in train dataset?
Ans
1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 510 out of  532 : 95.86466165413535
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [0]:

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

```
Number of Unique Variations : 1919
Truncating_Mutations    58
Deletion                52
Amplification           45
Fusions                 22
Overexpression           5
Q61L                     3
T58I                     3
R841K                    2
Y42C                     2
K117N                    2
Name: Variation, dtype: int64
```

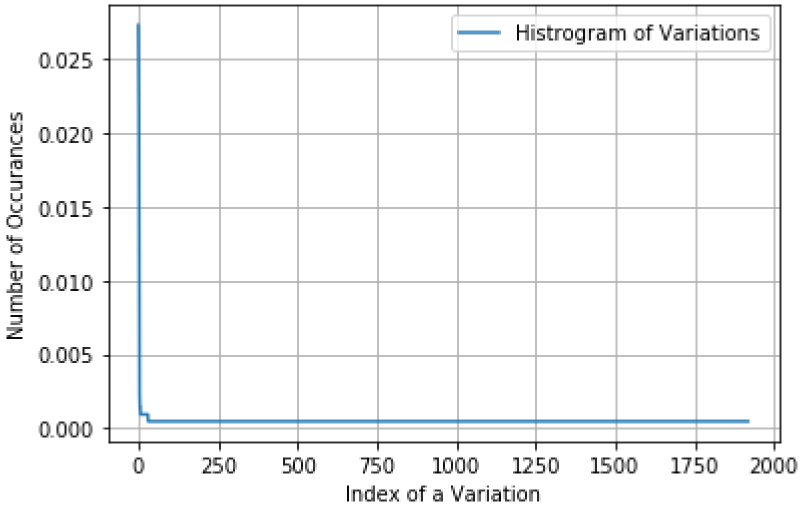
In [0]:

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

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

In [0]:

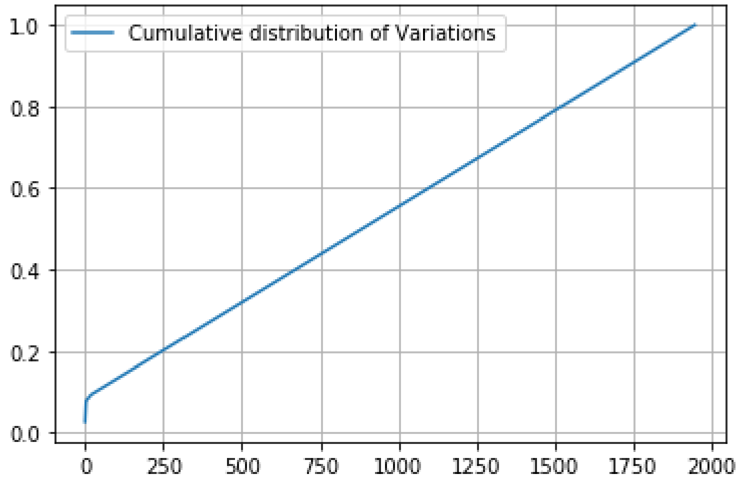
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [0]:

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

```
[0.02636535 0.04755179 0.06685499 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [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 = CountVectorizer()
variation_vectorizer = TfidfVectorizer(max_features=1000)
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. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

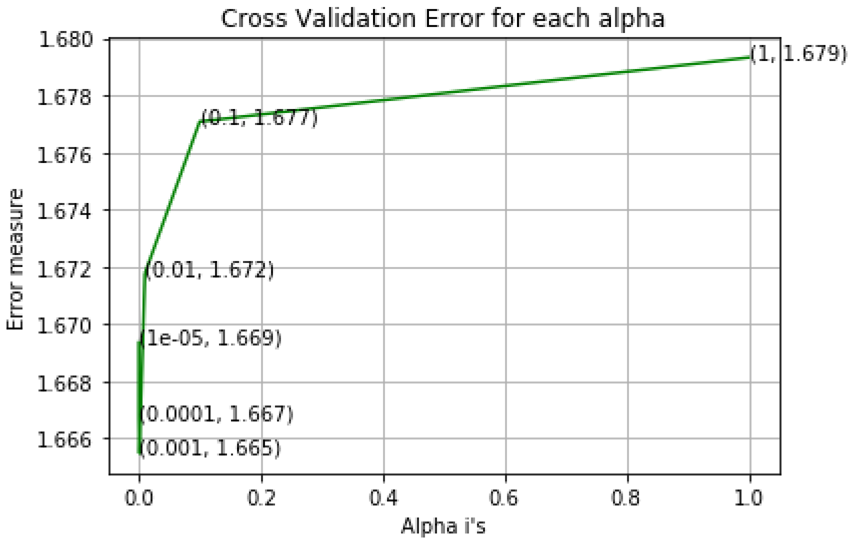
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1000)
```

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

Let's build a model just like the earlier!

```
In [0]: alpha = [10 ** x for x in range(-5, 1)]

        # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
        # ------------------------------
        # default parameters
        # SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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_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: 1.6693604111613807
For values of alpha =  0.0001 The log loss is: 1.6666919131949811
For values of alpha =  0.001 The log loss is: 1.6654784780089433
For values of alpha =  0.01 The log loss is: 1.6717616978641414
For values of alpha =  0.1 The log loss is: 1.6770824416792083
For values of alpha =  1 The log loss is: 1.679327717940895
```



```
For values of best alpha =  0.001 The train log loss is: 1.324320925208266
For values of best alpha =  0.001 The cross validation log loss is: 1.6654784780089433
For values of best alpha =  0.001 The test log loss is: 1.7336216343193875
```

In [10]:

```python
# Almost no improvement
table.add_row(['Var Feature One-Hot','Linear SVM', 1.73, 0.00])
print(table)
```

```
+----------------------+------------+---------------+-------------+
|      Vectorizer      |   Model    | Test Log Loss | Improvement |
+----------------------+------------+---------------+-------------+
|         NAN          |   Random   |      2.48     |      0      |
| Gene Feature One-Hot | Linear SVM |      1.19     |     0.01    |
| Var Feature One-Hot  | Linear SVM |      1.73     |     0.0     |
+----------------------+------------+---------------+-------------+
```

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]:

```python
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  1947  genes in test and cross validation data sets?
Ans
1. In test data 71 out of 665 : 10.676691729323307
2. In cross validation data 71 out of  532 : 13.345864661654137
```

3.2.3 Univariate Analysis on Text Feature

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

In [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 = CountVectorizer(min_df=3)
text_vectorizer = TfidfVectorizer(min_df=3, max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 1000
```

In [0]:

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [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))
        # For Tf-Idf vectorizer this becomes the tf-Idf scores
```

```
Counter({251.2522177607635: 1, 179.39874468978002: 1, 137.23988506274083: 1, 130.37912031654895: 1, 127.72575465869042: 1, 115.84189992691384: 1, 115.7565768066265: 1, 11
5.37865271191038: 1, 108.4666920935839: 1, 107.80489392384558: 1, 106.15590264332734: 1, 91.98072906721744: 1, 89.65636410795516: 1, 88.09461387649849: 1, 84.703141735537
73: 1, 83.74896574281752: 1, 81.45855668535646: 1, 78.9771390174896: 1, 76.91393644782916: 1, 76.86690511325695: 1, 76.47597573847696: 1, 75.52019595915498: 1, 71.9245717
7628181: 1, 69.7326961168429: 1, 68.23448084147601: 1, 67.57440950081453: 1, 66.05907707359563: 1, 65.53968131139166: 1, 65.37932899129838: 1, 63.842292874868605: 1, 63.7
9921604820528: 1, 63.42390707902485: 1, 62.123079554858904: 1, 58.38704467102912: 1, 58.15932785737887: 1, 58.130326021421915: 1, 56.40814391582066: 1, 56.25134614308504:
1, 55.82818636403047: 1, 53.69614995382579: 1, 50.925741900763406: 1, 50.66011645740224: 1, 48.81319205469113: 1, 48.29808705458015: 1, 47.34104308087886: 1, 46.122155964
01157: 1, 45.65253295132048: 1, 45.25991335611525: 1, 44.814213946272915: 1, 43.91763720678168: 1, 43.85572279435029: 1, 43.81909809804737: 1, 42.84846535510736: 1, 42.73
184097513837: 1, 42.672776994992155: 1, 42.35879196077927: 1, 42.21466430530964: 1, 41.99942843307959: 1, 41.96242259134214: 1, 41.93169852248974: 1, 41.74250028335423:
1, 41.539137136903584: 1, 41.526940835515575: 1, 41.40220255028021: 1, 40.81373148327584: 1, 40.78994634084748: 1, 40.033610672291104: 1, 39.7214540755789: 1, 38.89145317
670298: 1, 38.890042043294386: 1, 37.87408497353104: 1, 37.625379091153455: 1, 37.13901015330612: 1, 37.04822247921387: 1, 36.35735965463215: 1, 36.09041156242684: 1, 35.
83842611917853: 1, 35.65172903209027: 1, 35.648509992809984: 1, 35.62749050782718: 1, 35.476029136302195: 1, 35.409000122918584: 1, 35.404674869215974: 1, 35.277128283125
855: 1, 35.23344445331862: 1, 34.857321016216616: 1, 34.372211402296166: 1, 34.28554551368495: 1, 34.05418141170054: 1, 33.47833609990264: 1, 33.291821954159005: 1, 32.98
701030511611: 1, 32.419524851005754: 1, 32.217240425691045: 1, 32.164152576378164: 1, 32.10722117966565: 1, 32.02476583675842: 1, 32.019906863267046: 1, 31.9584270101236
3: 1, 31.74915726062845: 1, 31.668822616287017: 1, 31.57109713993643: 1, 31.47717298429625: 1, 31.339009831829614: 1, 31.288654441414153: 1, 31.2715561981l932: 1, 31.0898
46219583176: 1, 30.973224702682906: 1, 30.77139186119912: 1, 30.68616961536977: 1, 30.587242916329945: 1, 30.383713031250792: 1, 30.274968315368817: 1, 30.2525729421003З
6: 1, 30.100127352175374: 1, 29.909878912119883: 1, 29.90894954456297: 1, 29.750722016950917: 1, 29.646632651346675: 1, 29.51484207285247: 1, 29.206888310830145: 1, 29.08
0893023045213: 1, 29.077157330094373: 1, 29.05863921995519: 1, 28.9337867204075: 1, 28.66656295328302: 1, 28.582913272504776: 1, 28.273259485801994: 1, 28.26945473899569
4: 1, 27.950564630103727: 1, 27.949994159612125: 1, 27.846740838717636: 1, 27.550998383473527: 1, 27.364138453281175: 1, 27.332130247556: 1, 27.306474622571365: 1, 27.305
262620730392: 1, 27.193869601141987: 1, 26.96368535351023: 1, 26.89548904067164: 1, 26.88589170892986: 1, 26.87020914332724: 1, 26.612659519790203: 1, 26.611507608482853:
1, 26.45121561790403: 1, 26.445666946101912: 1, 26.42874899015342: 1, 26.428463588621895: 1, 25.988072539365636: 1, 25.78017205385859: 1, 25.662297075678993: 1, 25.595781
615645507: 1, 25.490236473429448: 1, 25.465133395526767: 1, 25.442303099601393: 1, 25.41095799708691: 1, 25.402373715407236: 1, 25.35729190337881: 1, 25.20936542262755:
1, 24.88220847930504: 1, 24.849527123443192: 1, 24.67108653115576: 1, 24.40142092027793: 1, 24.26411061638972: 1, 24.20490486244611: 1, 24.09784445876791: 1, 24.075790463
386795: 1, 24.061499943196694: 1, 24.043908004256444: 1, 24.009608404373594: 1, 23.983976716995645: 1, 23.931340556802432: 1, 23.887781197224584: 1, 23.887215912490607:
1, 23.715054904633924: 1, 23.709402542492718: 1, 23.702037763844952: 1, 23.685966117192244: 1, 23.635295267066: 1, 23.619901290144455: 1, 23.60504413316603: 1, 23.4620024
195946: 1, 23.298611879717363: 1, 23.25490828134861: 1, 23.17936349681295: 1, 23.13041920791293: 1, 23.08633413924704: 1, 22.881045870897907: 1, 22.53511283462085: 1, 22.
477792110789487: 1, 22.47305091489877: 1, 22.469040322078122: 1, 22.36723173435569: 1, 22.346629525202122: 1, 22.313178660897883: 1, 22.292544991687517: 1, 22.28025624613
2712: 1, 22.265775391723416: 1, 22.12248708103516: 1, 22.058952584323603: 1, 22.01041774477096: 1, 21.966917011274962: 1, 21.96002692424616: 1, 21.87950178742711: 1, 21.7
93687927551492: 1, 21.777636829576515: 1, 21.703350941452342: 1, 21.69944948665132: 1, 21.670060152899204: 1, 21.597105333880414: 1, 21.58605736489583: 1, 21.563260299850
615: 1, 21.537838334154078: 1, 21.44958959461668: 1, 21.410261875699298: 1, 21.261823462679775: 1, 21.256058620282005: 1, 21.24700203244903: 1, 21.228499577541864: 1, 21.
221969895274583: 1, 21.213306477480284: 1, 21.201736096260607: 1, 21.1210521506157: 1, 21.069636972789578: 1, 21.04422326574917: 1, 20.97671824187655: 1, 20.9582102133122
5: 1, 20.95164194055502: 1, 20.946218038422817: 1, 20.878605767851408: 1, 20.872483551385617: 1, 20.791936743260685: 1, 20.771163990452433: 1, 20.397490611948573: 1, 20.3
6105805426675: 1, 20.355150976801884: 1, 20.329992547467217: 1, 20.32973874570075: 1, 20.227736552177937: 1, 20.175251008454406: 1, 20.16341562092037: 1, 20.1554631892504
97: 1, 20.11630250088099: 1, 20.06080739799093: 1, 20.05839652037824: 1, 19.994420470538408: 1, 19.98656942325177: 1, 19.981739810051543: 1, 19.978137082708272: 1, 19.976
79415854508: 1, 19.85522291225902: 1, 19.82504309644599: 1, 19.815415238283652: 1, 19.747515612091973: 1, 19.743832943171164: 1, 19.739652432485595: 1, 19.73705217945086:
1, 19.721747156577116: 1, 19.596115650656753: 1, 19.576581821537452: 1, 19.560957790219536: 1, 19.43528391146967: 1, 19.320785650311823: 1, 19.29807691196644: 1, 19.25896
1450804367: 1, 19.258053730284953: 1, 19.21357797398974: 1, 19.145937982320866: 1, 19.10722359870803: 1, 19.099068342464804: 1, 19.044931721680406: 1, 19.038217012834437:
1, 19.019684080495484: 1, 19.00565912766754: 1, 18.998468242959646: 1, 18.95270145341245: 1, 18.95102016173014: 1, 18.929871575947846: 1, 18.923601031686644: 1, 18.915483
23756653: 1, 18.898316981447582: 1, 18.858369369150633: 1, 18.79892893169373: 1, 18.70415046579018: 1, 18.691734897009812: 1, 18.63606587140689: 1, 18.588311692333924: 1,
18.57210347391099: 1, 18.429136491936866: 1, 18.425358077540455: 1, 18.42070908390541: 1, 18.373790008286292: 1, 18.32653804592984: 1, 18.2552359536696: 1, 18.23540238807
48: 1, 18.176674897472502: 1, 18.155515215490606: 1, 18.032895211575685: 1, 18.03093201900719: 1, 17.995725122489343: 1, 17.969238425727657: 1, 17.95427133921421: 1, 17.9
28529451094963: 1, 17.905495274896197: 1, 17.868459187693418: 1, 17.85162822130295: 1, 17.835313650036653: 1, 17.828292787136487: 1, 17.753522740824508: 1, 17.74536265249
7487: 1, 17.66150602795946: 1, 17.63999131104508: 1, 17.584189374610002: 1, 17.545681182179738: 1, 17.533288504126933: 1, 17.50256514455262: 1, 17.477173414425177: 1, 17.
459504289588818: 1, 17.455227086134478: 1, 17.451875321732498: 1, 17.429230716994923: 1, 17.414409629570766: 1, 17.401045796201384: 1, 17.379792662467764: 1, 17.360134176
006238: 1, 17.34712260987314: 1, 17.343548606188584: 1, 17.312287536197065: 1, 17.260937139550084: 1, 17.250042632888533: 1, 17.215158317911502: 1, 17.183451574654082: 1,
17.183249721997758: 1, 17.17857965684129: 1, 17.146414559469633: 1, 17.13570161529653: 1, 17.133663068654833: 1, 17.11493366416714: 1, 17.11319228574685: 1, 17.0668219569
14005: 1, 17.062614772587736: 1, 17.043109435512616: 1, 17.041925089162874: 1, 17.032150911628545: 1, 16.972496851600397: 1, 16.930353776868788: 1, 16.89670333972397: 1,
16.889812607091912: 1, 16.81037152652856: 1, 16.78843423478995: 1, 16.669181354917757: 1, 16.642438037648635: 1, 16.64157207119232: 1, 16.60583966749821: 1, 16.5476051352
49356: 1, 16.544296608759208: 1, 16.505078123429698: 1, 16.503476344086334: 1, 16.4902075632775: 1, 16.478878918519893: 1, 16.442046870875863: 1, 16.368758446176212: 1, 1
6.339712454823644: 1, 16.311325640606828: 1, 16.31067336438338: 1, 16.23945561536795: 1, 16.231725415264528: 1, 16.14949586152229: 1, 16.11463158912284: 1, 16.09397545478
608: 1, 16.036651606834614: 1, 15.990276392850875: 1, 15.967485640975049: 1, 15.91766651042907: 1, 15.916576124442681: 1, 15.895229671718194: 1, 15.878864277835696: 1, 1
5.872102062136113: 1, 15.868974300740716: 1, 15.823253649567299: 1, 15.765927042044392: 1, 15.76104907893119: 1, 15.714880628800618: 1, 15.704255032750726: 1, 15.63779810
8570093: 1, 15.636997378984725: 1, 15.5962376561776: 1, 15.573351897807854: 1, 15.557798922700158: 1, 15.553814435010429: 1, 15.543975593861685: 1, 15.533518050186071: 1,
15.52818259169806: 1, 15.48911053541455: 1, 15.375786369327548: 1, 15.3471426992849: 1, 15.319676965142088: 1, 15.264474260589276: 1, 15.188669943765595: 1, 15.1863834633
4823: 1, 15.15234007661712: 1, 15.115586748323883: 1, 15.103248020740066: 1, 15.097275038526528: 1, 15.058246500827016: 1, 15.049548788439072: 1, 15.035960964755859: 1, 1
4.972180582059135: 1, 14.948995392867072: 1, 14.925902000158722: 1, 14.908856720292796: 1, 14.899181395166076: 1, 14.844467197416812: 1, 14.785177196446393: 1, 14.7443355
19005116: 1, 14.733503797579166: 1, 14.724014386648566: 1, 14.722405205922517: 1, 14.719501971550923: 1, 14.717135414800758: 1, 14.706093591733826: 1, 14.683648821963029:
1, 14.676584537759172: 1, 14.675398499144379: 1, 14.66898148079602: 1, 14.653898545018942: 1, 14.629800644239998: 1, 14.621715402650347: 1, 14.608750573112982: 1, 14.5747
1026600431: 1, 14.564596259795147: 1, 14.560964580132673: 1, 14.555190037761408: 1, 14.540540210157273: 1, 14.514457659763714: 1, 14.505783229271094: 1, 14.47369458892462
8: 1, 14.4327906940325: 1, 14.4152305850901: 1, 14.41285875254492: 1, 14.39444104969646: 1, 14.360403293711494: 1, 14.281102876759531: 1, 14.278249931351555: 1, 14.248635
17873059: 1, 14.236034719388098: 1, 14.218059694829122: 1, 14.214880160803768: 1, 14.193837423168695: 1, 14.185284789046309: 1, 14.17904161444134: 1, 14.164630703581677:
1, 14.153488792206652: 1, 14.144469406005364: 1, 14.120438389203624: 1, 14.11048893928829: 1, 14.106215432471236: 1, 14.09909771485471: 1, 14.059713262895908: 1, 14.04411
6891720046: 1, 14.029311266571836: 1, 14.008724852355561: 1, 13.971077185018489: 1, 13.96067494129226: 1, 13.922754476031198: 1, 13.887333485034137: 1, 13.87338961594312:
1, 13.841759847727362: 1, 13.77231886011738: 1, 13.758131489016908: 1, 13.614079491497172: 1, 13.60921206494711: 1, 13.599384872072003: 1, 13.595564939301486: 1, 13.55230
602669988: 1, 13.550875818956852: 1, 13.523394194643732: 1, 13.516331121961546: 1, 13.498478482708947: 1, 13.498011671768932: 1, 13.493505640160638: 1, 13.49134031539306
7: 1, 13.47885244466274: 1, 13.461541124201172: 1, 13.447265029204019: 1, 13.444479335470874: 1, 13.369691441508095: 1, 13.368732448310842: 1, 13.365964190887343: 1, 13.3
```

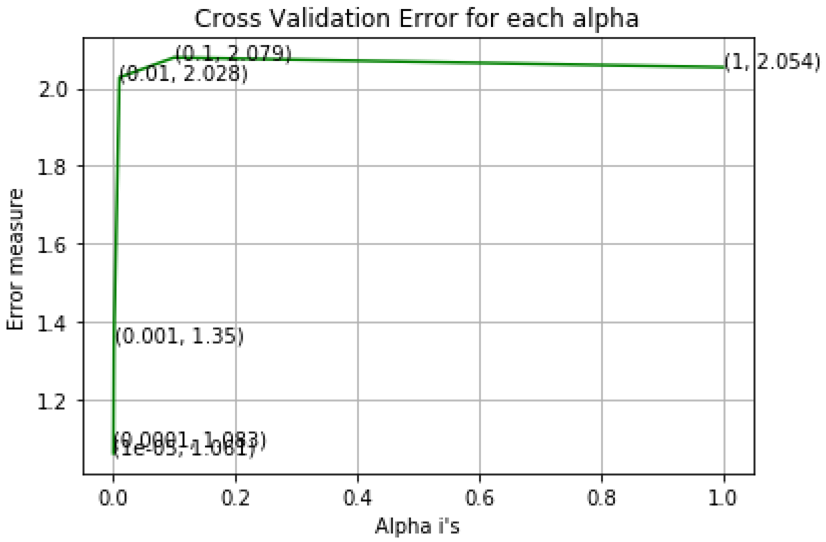
```
2794062222218: 1, 13.261735912601406: 1, 13.228033500956721: 1, 13.183450049667572: 1, 13.157304651826948: 1, 13.154035243574342: 1, 13.114810681797538: 1, 13.10633729892
2478: 1, 13.096523624204236: 1, 13.080865057820452: 1, 13.030836943767344: 1, 13.017407661447384: 1, 13.002661377819244: 1, 12.99233767849999: 1, 12.944970917999191: 1, 1
2.94144843130449: 1, 12.927146413407725: 1, 12.909253321712951: 1, 12.898267292961044: 1, 12.893075996488614: 1, 12.888626960592456: 1, 12.836480124906664: 1, 12.79644371
7893556: 1, 12.793685247276443: 1, 12.792519158360976: 1, 12.75466334499257: 1, 12.743309987492124: 1, 12.720990084888841: 1, 12.708769284623042: 1, 12.687448615945447:
1, 12.668982188360095: 1, 12.645393920520327: 1, 12.631598817407246: 1, 12.623692231036582: 1, 12.617755925073363: 1, 12.607278615961304: 1, 12.575839560707285: 1, 12.563
531632446995: 1, 12.561223714528975: 1, 12.561058696233834: 1, 12.556289931283358: 1, 12.547259404525509: 1, 12.530535501048591: 1, 12.524008888471924: 1, 12.512382994331
476: 1, 12.483835416580426: 1, 12.455578617795576: 1, 12.450063449011012: 1, 12.43883680332174: 1, 12.434829523497221: 1, 12.423063745554684: 1, 12.37603036258224: 1, 12.
343444393298297: 1, 12.340721623927344: 1, 12.328684821034859: 1, 12.30249868522257: 1, 12.284733314064393: 1, 12.27241175299284: 1, 12.269249474259711: 1, 12.21893223649
9525: 1, 12.198554472144794: 1, 12.19340429179947: 1, 12.192564942016038: 1, 12.17259663810287: 1, 12.142739967451687: 1, 12.130296692083482: 1, 12.083531193655379: 1, 1
2.050089642327901: 1, 12.043625746091351: 1, 12.027480236022955: 1, 12.013146399274623: 1, 11.994140190143966: 1, 11.980710109327461: 1, 11.943936605634887: 1, 11.9202836
99136723: 1, 11.919559906206597: 1, 11.901481047740544: 1, 11.872707032218266: 1, 11.852535928476449: 1, 11.850797239324992: 1, 11.823672300294243: 1, 11.809781188781429:
1, 11.804656168321014: 1, 11.800450891696771: 1, 11.798726422912031: 1, 11.79691942230899: 1, 11.79056740244767: 1, 11.766008872825468: 1, 11.762774895430997: 1, 11.74181
4088800888: 1, 11.738593635531: 1, 11.73830665933832: 1, 11.737847159833002: 1, 11.720904447233437: 1, 11.700675698007458: 1, 11.685275835952257: 1, 11.652686394740728:
1, 11.648804165857378: 1, 11.64709339079301: 1, 11.646099109454436: 1, 11.616050964165224: 1, 11.594582629013397: 1, 11.576645765380205: 1, 11.551048762791574: 1, 11.5219
71581802944: 1, 11.52061952752484: 1, 11.477439512551614: 1, 11.457206756869242: 1, 11.450931002876265: 1, 11.449460565969131: 1, 11.428931253095874: 1, 11.42592029859511
3: 1, 11.419298418936899: 1, 11.410340332170305: 1, 11.398038667627537: 1, 11.394293463078395: 1, 11.371765622315243: 1, 11.344603033305917: 1, 11.332544562232359: 1, 11.
33240692404704: 1, 11.33079451710559: 1, 11.294853130246988: 1, 11.294147668395428: 1, 11.265782537520755: 1, 11.260793463414402: 1, 11.245781438946413: 1, 11.22420580553
2844: 1, 11.17548570532491: 1, 11.171011125393573: 1, 11.165123098819475: 1, 11.164818967224962: 1, 11.145611371940362: 1, 11.144389795704436: 1, 11.134593971831022: 1, 1
1.132286033106508: 1, 11.12524657018567: 1, 11.107027785103007: 1, 11.091556318560475: 1, 11.063288096046904: 1, 11.042236446833034: 1, 11.041685138704892: 1, 11.03533559
7822188: 1, 11.03055483251462: 1, 11.027669509667776: 1, 10.988914226303208: 1, 10.983123026309688: 1, 10.9798637741982: 1, 10.969383890310718: 1, 10.925114902411101: 1,
10.90861260446834: 1, 10.891620761454686: 1, 10.890540466801644: 1, 10.876869454545972: 1, 10.87492111233909: 1, 10.849998626114989: 1, 10.837433173014484: 1, 10.83474194
846634: 1, 10.830412449185259: 1, 10.829916450960447: 1, 10.816232214891038: 1, 10.81339904293614: 1, 10.79598968176833: 1, 10.78930472343356: 1, 10.7617380022807: 1, 1
0.755757156001001: 1, 10.732899900914418: 1, 10.724944026141722: 1, 10.719304509217276: 1, 10.68915052876294: 1, 10.68756790521067: 1, 10.677019467252071: 1, 10.672620799
940727: 1, 10.666976977940239: 1, 10.658609118791583: 1, 10.656350963108453: 1, 10.624101709172464: 1, 10.603304416142405: 1, 10.58134823738523: 1, 10.580541211619396: 1,
10.56137907313706: 1, 10.547852226597662: 1, 10.532876421948966: 1, 10.51457887380899: 1, 10.493519653354443: 1, 10.481062930449042: 1, 10.44008346454527: 1, 10.425761189
460813: 1, 10.411875203569613: 1, 10.393792885992879: 1, 10.387140373909824: 1, 10.37999787566731: 1, 10.374822666602196: 1, 10.373372377824612: 1, 10.370948368695986: 1,
10.361834731045246: 1, 10.317910756862505: 1, 10.31734759703663: 1, 10.276954390859155: 1, 10.266223481134533: 1, 10.246344024655578: 1, 10.240100936310128: 1, 10.2376339
38262361: 1, 10.226741628060811: 1, 10.225897911667392: 1, 10.22324986116367: 1, 10.212659144138177: 1, 10.200570150149394: 1, 10.198398679986205: 1, 10.17530472668758:
1, 10.167002671839517: 1, 10.15611713832785: 1, 10.149619663859742: 1, 10.14915399803771: 1, 10.136519183988943: 1, 10.109989852357042: 1, 10.104950236637334: 1, 10.09371
5658425486: 1, 10.053526631558544: 1, 10.034365422302846: 1, 10.01424668356317: 1, 10.010861135493332: 1, 9.997281364403753: 1, 9.994869143177372: 1, 9.979075467851807:
1, 9.964407180915074: 1, 9.93847450276803: 1, 9.934720636596923: 1, 9.934673248900888: 1, 9.888989874930637: 1, 9.88788148868648: 1, 9.877915444198056: 1, 9.8665993910141
3: 1, 9.857194734789962: 1, 9.848970663006723: 1, 9.84427891204719: 1, 9.83246020306723: 1, 9.828418742160531: 1, 9.822431460545534: 1, 9.816938183355585: 1, 9.8141533099
34348: 1, 9.80230441133057: 1, 9.802161016606943: 1, 9.796451136336072: 1, 9.792562957064403: 1, 9.790242764978398: 1, 9.78102137141489: 1, 9.766461383542733: 1, 9.756998
225903503: 1, 9.74703291569974: 1, 9.744212849509724: 1, 9.738302547973813: 1, 9.71816818946839: 1, 9.686688645596968: 1, 9.675534694393734: 1, 9.6674425828773: 1, 9.6581
70957593786: 1, 9.656972978050597: 1, 9.645664369562827: 1, 9.62727545951156: 1, 9.620326575889663: 1, 9.617485507925414: 1, 9.610104798374255: 1, 9.58956504708696: 1, 9.
589053185505534: 1, 9.581264403474208: 1, 9.578545989147909: 1, 9.57063386012378: 1, 9.508717099170843: 1, 9.496838119871406: 1, 9.475497122818243: 1, 9.46853810536491:
1, 9.46240133544092: 1, 9.453751240426259: 1, 9.453384723565298: 1, 9.439181957053483: 1, 9.429147689553357: 1, 9.428443368620886: 1, 9.423934521088475: 1, 9.416522681449
532: 1, 9.414630028312473: 1, 9.393733070815587: 1, 9.38796931975765: 1, 9.376748875810382: 1, 9.328785229014144: 1, 9.326231323958165: 1, 9.325919729947122: 1, 9.3162796
16552947: 1, 9.312491862683325: 1, 9.298998186378352: 1, 9.296787627887271: 1, 9.282941794810897: 1, 9.281989964150071: 1, 9.258916458937342: 1, 9.255122976163038: 1, 9.2
52957197413277: 1, 9.232524598533862: 1, 9.23086495619827: 1, 9.21514538367103: 1, 9.192881414225198: 1, 9.18376735326325: 1, 9.181338692939477: 1, 9.173874628095762: 1,
9.17058804247006: 1, 9.158833742842402: 1, 9.158728315386007: 1, 9.144561492957202: 1, 9.143754818993571: 1, 9.13577784340806: 1, 9.134608268251906: 1, 9.103800537501373:
1, 9.075989969802196: 1, 9.074028752840402: 1, 9.071125644184946: 1, 9.068101379236976: 1, 9.063768547411295: 1, 9.063404833284995: 1, 9.053890726103909: 1, 9.05310570750
1295: 1, 9.051820293861338: 1, 9.04813027008085: 1, 9.038887625229671: 1, 9.038674082475445: 1, 9.02853918708622: 1, 8.996736994045738: 1, 8.987636408467974: 1, 8.9811625
71387847: 1, 8.96487156689383: 1, 8.963124656952074: 1, 8.961380421432308: 1, 8.961203544398444: 1, 8.957196967631749: 1, 8.95053945034118: 1, 8.941965055506927: 1, 8.931
684606148618: 1, 8.925052803187693: 1, 8.919372814651108: 1, 8.908612189711345: 1, 8.899699156739828: 1, 8.877621441601354: 1, 8.868399251066187: 1, 8.865869014548403: 1,
8.861580254518362: 1, 8.85002982392078: 1, 8.846472163083222: 1, 8.837536810234134: 1, 8.827963862511242: 1, 8.818060175133985: 1, 8.81230382375564: 1, 8.799246897003778:
1, 8.791532137423326: 1, 8.788091926803618: 1, 8.782457324415299: 1, 8.75164535346011: 1, 8.745548119688138: 1, 8.729692925306816: 1, 8.724241625493432: 1, 8.723572639367
532: 1, 8.723017731996537: 1, 8.712679824013135: 1, 8.699334766553138: 1, 8.687387455448722: 1, 8.684534897934208: 1, 8.683203277513556: 1, 8.678751440683712: 1, 8.661241
345685816: 1, 8.641122777713793: 1, 8.62383866108619: 1, 8.58910793375445: 1, 8.583477868325508: 1, 8.573035186410738: 1, 8.559349083074355: 1, 8.534781948804914: 1, 8.52
9305014960915: 1, 8.517676063464537: 1, 8.515548645539216: 1, 8.5001064133601: 1, 8.48877496988269: 1, 8.48026084255252: 1, 8.473944078368968: 1, 8.47236207787773: 1, 8.4
4161691465946: 1, 8.437728235470855: 1, 8.398463348819979: 1, 8.382151763061982: 1, 8.379397651407633: 1, 8.37899877345558: 1, 8.371618431618236: 1, 8.371469921257512: 1,
8.370266578559532: 1, 8.360215544741248: 1, 8.343972349181149: 1, 8.334538651253883: 1, 8.333328271193935: 1, 8.329134013896894: 1, 8.327818026844556: 1, 8.31830626820694
8: 1, 8.31475128547273: 1, 8.297337885243431: 1, 8.278941758484903: 1, 8.26378132227445: 1, 8.246956587716491: 1, 8.243821638169253: 1, 8.241868638124103: 1, 8.2264527561
33696: 1, 8.214567990370421: 1, 8.210498601791109: 1, 8.19585021733768: 1, 8.189639851330048: 1, 8.18611089492415: 1, 8.180078699670329: 1, 8.151552669383861: 1, 8.150323
474756439: 1, 8.125007056148055: 1, 8.101398203617284: 1, 8.087337652003894: 1, 8.07034005404388: 1, 8.068495694979651: 1, 8.038695413809787: 1, 7.9905791687145795: 1, 7.
985909180232167: 1, 7.958980060843109: 1, 7.946670728886254: 1, 7.9318861404471175: 1, 7.920693000734725: 1, 7.917456294481619: 1, 7.914694232188179: 1, 7.89517597690287
2: 1, 7.889072425326686: 1, 7.880706654670192: 1, 7.879114641694707: 1, 7.877753032163325: 1, 7.874498667272461: 1, 7.869305113045058: 1, 7.867096457413153: 1, 7.85009679
8483834: 1, 7.81067299763914: 1, 7.798952351410952: 1, 7.795633558938193: 1, 7.793340416429904: 1, 7.789733859025618: 1, 7.779465957026187: 1, 7.777773385703349: 1, 7.739
4792890445885: 1, 7.736419993908609: 1, 7.729303497180483: 1, 7.7269441480640575: 1, 7.702199060180337: 1, 7.696568241030091: 1, 7.692760285514637: 1, 7.6772533541705466:
1, 7.663909872230474: 1, 7.652836500141612: 1, 7.645959813470575: 1, 7.623528739692183: 1, 7.61487532810329: 1, 7.609219393063418: 1, 7.603916586643343: 1, 7.602194103065
705: 1, 7.600881611756667: 1, 7.59922904411732: 1, 7.577175323655055: 1, 7.572487603854625: 1, 7.529109128265301: 1, 7.525513891560966: 1, 7.519444155338969: 1, 7.5124261
71040218: 1, 7.45495915042407: 1, 7.454946772554533: 1, 7.448973892110388: 1, 7.44193645453784: 1, 7.441024575406907: 1, 7.412266493890808: 1, 7.380837055393127: 1, 7.369
90121694327: 1, 7.354494903489863: 1, 7.311924524499399: 1, 7.277811686904291: 1, 7.256661534951594: 1, 7.237745650918049: 1, 7.201365559699845: 1, 7.188383587987079: 1,
7.131623982417213: 1, 7.10271994044156: 1, 7.099802212258368: 1, 7.072526255904265: 1, 7.059748570762206: 1, 7.034340912640289: 1, 7.033364114219887: 1, 7.03212486029524
3: 1, 7.019622535374786: 1, 6.978228307175894: 1, 6.934185759236185: 1, 6.89684157865458: 1, 6.882518622180837: 1, 6.852181935676014: 1, 6.836591498481608: 1, 6.825135728
```

```
281795: 1, 6.818753737586301: 1, 6.790163438032933: 1, 6.785422847869152: 1, 6.780074269557475: 1, 6.7772827025202: 1, 6.685247167012078: 1, 6.670819411137737: 1, 6.66243
7942189571: 1, 6.654394656303248: 1, 6.548233946331805: 1, 6.436078094679334: 1, 6.429948083698248: 1, 6.412353153290557: 1, 6.377839934381468: 1})
```

In [0]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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: 1.0608355031186114
For values of alpha =  0.0001 The log loss is: 1.0831888085260073
For values of alpha =  0.001 The log loss is: 1.3495324868213179
For values of alpha =  0.01 The log loss is: 2.0278921974471555
For values of alpha =  0.1 The log loss is: 2.0792015970662163
For values of alpha =  1 The log loss is: 2.0541225350722345
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7467621092781309
For values of best alpha =  1e-05 The cross validation log loss is: 1.0608355031186114
For values of best alpha =  1e-05 The test log loss is: 1.1051700479594095
```

In [11]:
```
# Test log loss improvement - 0.09
table.add_row(['Text Feature One-Hot','Linear SVM', 1.10, 0.09])
print(table)
```

```
+----------------------+------------+---------------+-------------+
|      Vectorizer      |   Model    | Test Log Loss | Improvement |
+----------------------+------------+---------------+-------------+
|         NAN          |   Random   |      2.48     |      0      |
| Gene Feature One-Hot | Linear SVM |      1.19     |     0.01    |
| Var Feature One-Hot  | Linear SVM |      1.73     |     0.0     |
| Text Feature One-Hot | Linear SVM |      1.1      |     0.09    |
+----------------------+------------+---------------+-------------+
```

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_vec = TfidfVectorizer(min_df=3, max_features=1000)
    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")
```

```
95.4 % of word of test data appeared in train data
93.4 % 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 willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [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_impfeature_names(indices, text, gene, var, no_features):
    # gene_count_vec = CountVectorizer()
    # var_count_vec = CountVectorizer()
    # text_count_vec = CountVectorizer(min_df=3)
    gene_count_vec = TfidfVectorizer(max_features=1000)
    var_count_vec = TfidfVectorizer(max_features=1000)
    text_count_vec = TfidfVectorizer(min_df=3, max_features=1000)

    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]]

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, 2226)
(number of data points * number of features) in test data =  (665, 2226)
(number of data points * number of features) in cross validation data = (532, 2226)
```

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

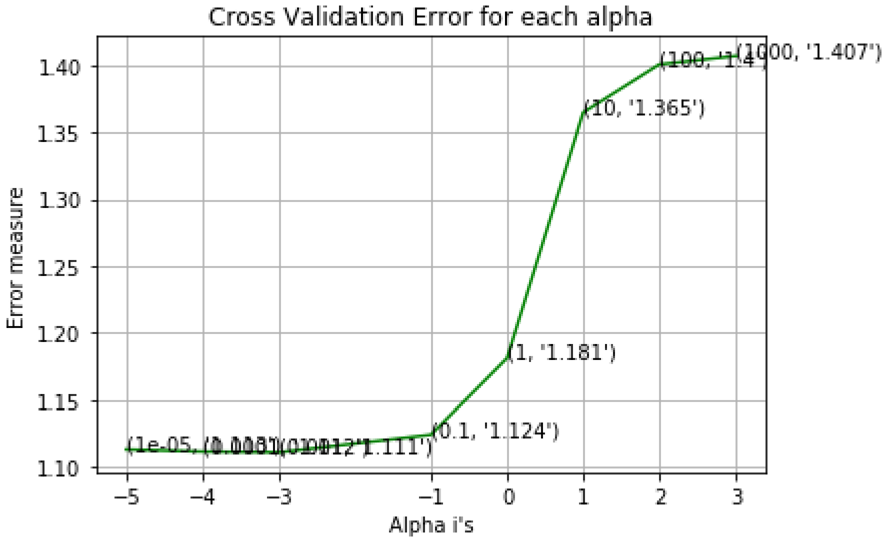
4.1.1.1. Hyper parameter tuning

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=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 : 1.11317840390721
for alpha = 0.0001
Log Loss : 1.1116122972134819
for alpha = 0.001
Log Loss : 1.1110256351459618
for alpha = 0.1
Log Loss : 1.12411084408832
for alpha = 1
Log Loss : 1.1812091490690204
for alpha = 10
Log Loss : 1.3645041614436861
for alpha = 100
Log Loss : 1.400407319282646
for alpha = 1000
Log Loss : 1.4067251806311325
```



```
For values of best alpha =  0.001 The train log loss is: 0.757654987325886
For values of best alpha =  0.001 The cross validation log loss is: 1.1110256351459618
For values of best alpha =  0.001 The test log loss is: 1.1776872664354325
```

In [12]:
```
# Test log improvement - 0.1
table.add_row(['TfIdf All Features','Naive Bayes', 1.17, 0.1])
print(table)
```

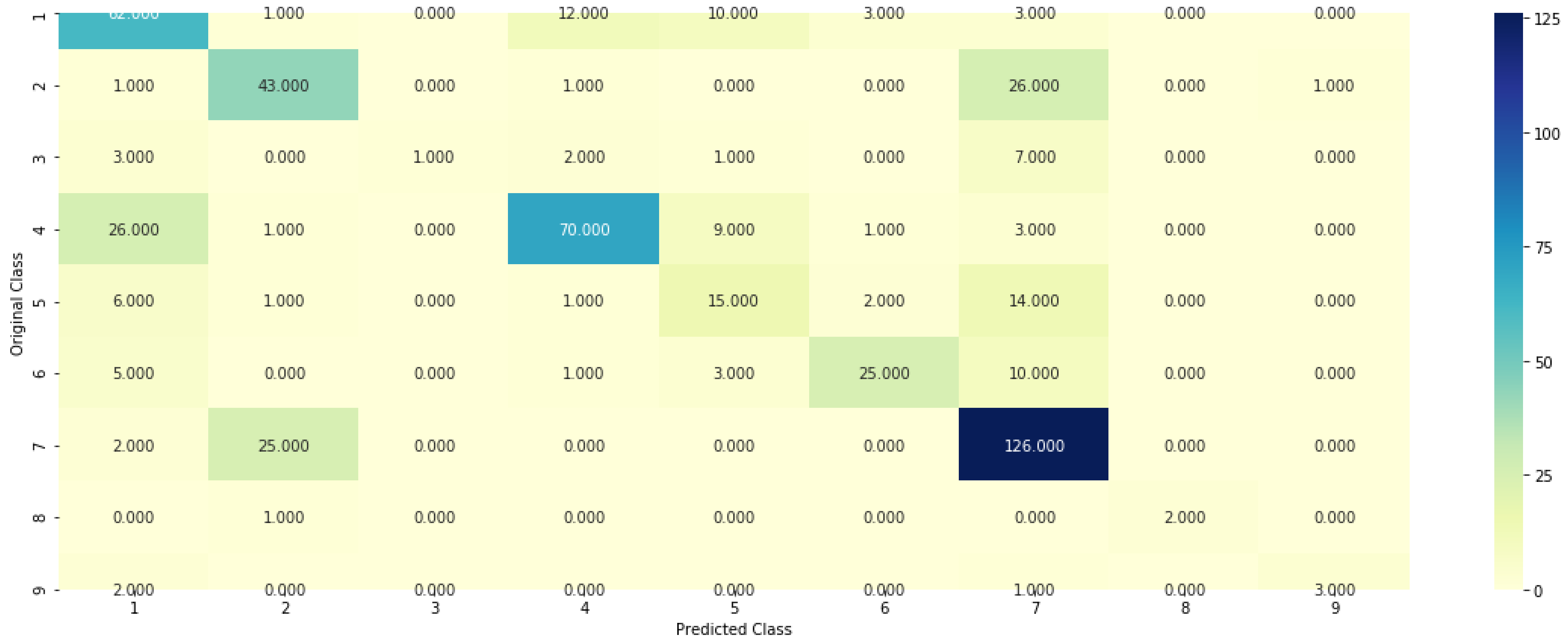
```
+----------------------+-------------+---------------+-------------+
|      Vectorizer      |    Model    | Test Log Loss | Improvement |
+----------------------+-------------+---------------+-------------+
|         NAN          |    Random   |      2.48     |      0      |
| Gene Feature One-Hot | Linear SVM  |      1.19     |     0.01    |
| Var Feature One-Hot  | Linear SVM  |      1.73     |     0.0     |
| Text Feature One-Hot | Linear SVM  |      1.1      |     0.09    |
|  TfIdf All Features  | Naive Bayes |      1.17     |     0.1     |
+----------------------+-------------+---------------+-------------+
```

4.1.1.2. Testing the model with best hyper paramters

In [0]:

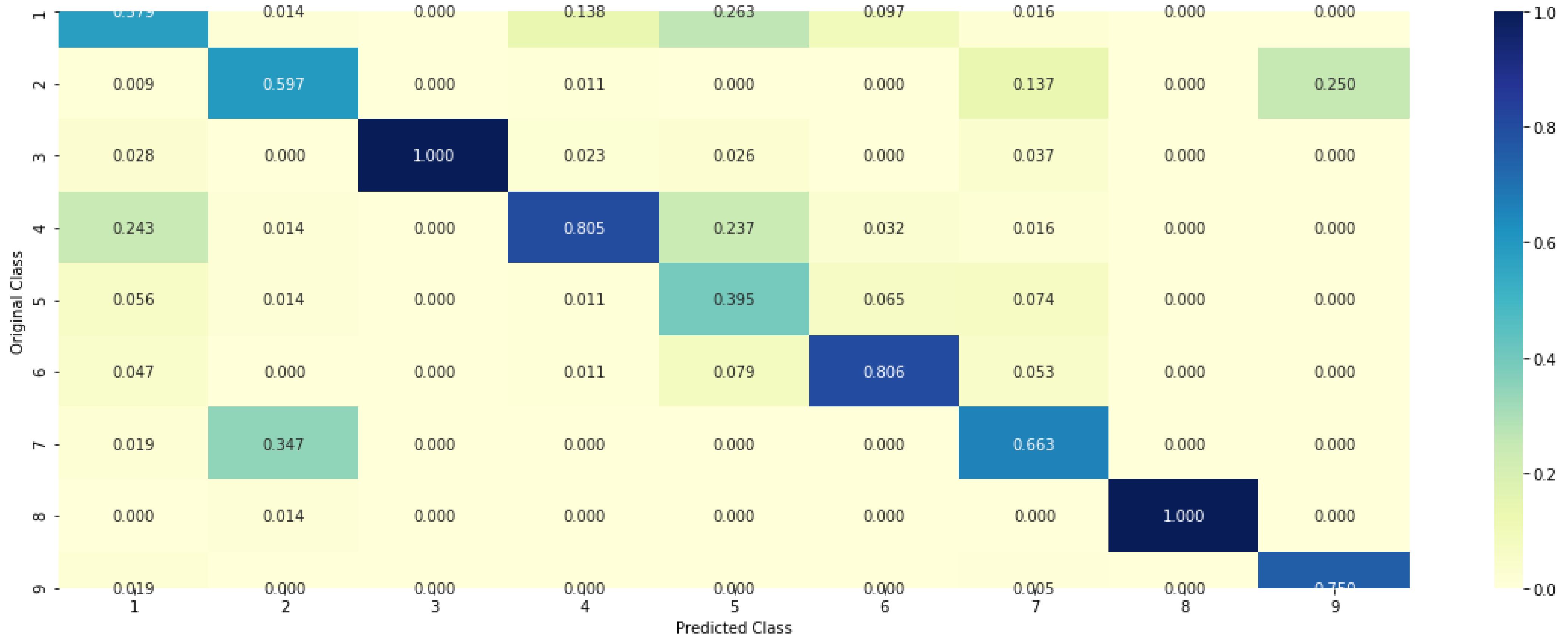
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.1110256351459618
Number of missclassified point : 0.34774436090225563
-------------------- Confusion matrix --------------------
```



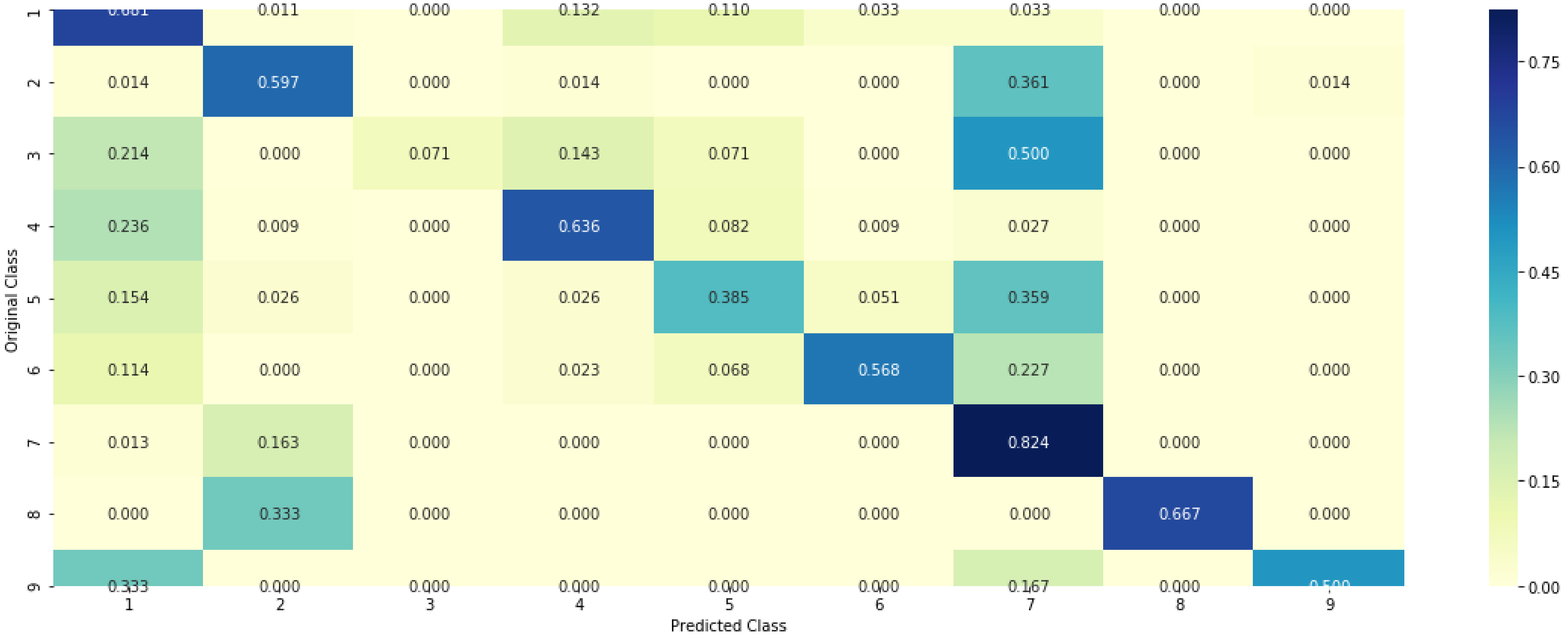
	1	2	3	4	5	6	7	8	9
1	62.000	1.000	0.000	12.000	10.000	3.000	3.000	0.000	0.000
2	1.000	43.000	0.000	1.000	0.000	0.000	26.000	0.000	1.000
3	3.000	0.000	1.000	2.000	1.000	0.000	7.000	0.000	0.000
4	26.000	1.000	0.000	70.000	9.000	1.000	3.000	0.000	0.000
5	6.000	1.000	0.000	1.000	15.000	2.000	14.000	0.000	0.000
6	5.000	0.000	0.000	1.000	3.000	25.000	10.000	0.000	0.000
7	2.000	25.000	0.000	0.000	0.000	0.000	126.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000
9	2.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	3.000

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



	1	2	3	4	5	6	7	8	9
1	0.579	0.014	0.000	0.138	0.263	0.097	0.016	0.000	0.000
2	0.009	0.597	0.000	0.011	0.000	0.000	0.137	0.000	0.250
3	0.028	0.000	1.000	0.023	0.026	0.000	0.037	0.000	0.000
4	0.243	0.014	0.000	0.805	0.237	0.032	0.016	0.000	0.000
5	0.056	0.014	0.000	0.011	0.395	0.065	0.074	0.000	0.000
6	0.047	0.000	0.000	0.011	0.079	0.806	0.053	0.000	0.000
7	0.019	0.347	0.000	0.000	0.000	0.000	0.663	0.000	0.000
8	0.000	0.014	0.000	0.000	0.000	0.000	0.000	1.000	0.000
9	0.019	0.000	0.000	0.000	0.000	0.000	0.005	0.000	0.750

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



4.1.1.3. Feature Importance, Correctly classified point

In [0]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0587 0.0525 0.0164 0.7391 0.033  0.0315 0.0637 0.0028 0.0021]]
Actual Class : 4
--------------------------------------------------
8 Text feature [activity] present in test data point [True]
11 Text feature [protein] present in test data point [True]
13 Text feature [proteins] present in test data point [True]
17 Text feature [missense] present in test data point [True]
23 Text feature [whereas] present in test data point [True]
27 Text feature [two] present in test data point [True]
33 Text feature [mutations] present in test data point [True]
35 Text feature [catalytic] present in test data point [True]
36 Text feature [suppressor] present in test data point [True]
37 Text feature [described] present in test data point [True]
39 Text feature [may] present in test data point [True]
51 Text feature [previously] present in test data point [True]
53 Text feature [vitro] present in test data point [True]
74 Text feature [ability] present in test data point [True]
79 Text feature [mutation] present in test data point [True]
83 Text feature [suggesting] present in test data point [True]
84 Text feature [cells] present in test data point [True]
85 Text feature [contribute] present in test data point [True]
86 Text feature [effect] present in test data point [True]
92 Text feature [could] present in test data point [True]
Out of the top  100  features  20 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [0]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0593 0.055  0.0172 0.0595 0.0347 0.0326 0.7365 0.0029 0.0022]]
Actual Class : 7
--------------------------------------------------
15 Text feature [activation] present in test data point [True]
16 Text feature [kinase] present in test data point [True]
17 Text feature [activated] present in test data point [True]
18 Text feature [downstream] present in test data point [True]
20 Text feature [inhibitor] present in test data point [True]
21 Text feature [cells] present in test data point [True]
22 Text feature [expressing] present in test data point [True]
23 Text feature [presence] present in test data point [True]
24 Text feature [signaling] present in test data point [True]
25 Text feature [inhibitors] present in test data point [True]
26 Text feature [however] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [independent] present in test data point [True]
29 Text feature [10] present in test data point [True]
30 Text feature [sensitive] present in test data point [True]
32 Text feature [constitutive] present in test data point [True]
33 Text feature [treatment] present in test data point [True]
38 Text feature [activating] present in test data point [True]
39 Text feature [mutations] present in test data point [True]
40 Text feature [growth] present in test data point [True]
41 Text feature [inhibition] present in test data point [True]
42 Text feature [previously] present in test data point [True]
43 Text feature [factor] present in test data point [True]
44 Text feature [shown] present in test data point [True]
45 Text feature [compared] present in test data point [True]
47 Text feature [treated] present in test data point [True]
49 Text feature [well] present in test data point [True]
50 Text feature [addition] present in test data point [True]
51 Text feature [cell] present in test data point [True]
52 Text feature [increased] present in test data point [True]
53 Text feature [phosphorylation] present in test data point [True]
54 Text feature [recently] present in test data point [True]
55 Text feature [found] present in test data point [True]
57 Text feature [may] present in test data point [True]
58 Text feature [similar] present in test data point [True]
59 Text feature [activate] present in test data point [True]
60 Text feature [3b] present in test data point [True]
61 Text feature [survival] present in test data point [True]
62 Text feature [concentrations] present in test data point [True]
63 Text feature [potential] present in test data point [True]
64 Text feature [enhanced] present in test data point [True]
65 Text feature [tyrosine] present in test data point [True]
66 Text feature [constitutively] present in test data point [True]
68 Text feature [showed] present in test data point [True]
69 Text feature [absence] present in test data point [True]
70 Text feature [mutation] present in test data point [True]
71 Text feature [mutant] present in test data point [True]
72 Text feature [total] present in test data point [True]
74 Text feature [20] present in test data point [True]
75 Text feature [without] present in test data point [True]
77 Text feature [receptor] present in test data point [True]
78 Text feature [oncogenic] present in test data point [True]
81 Text feature [suggest] present in test data point [True]
83 Text feature [3a] present in test data point [True]
84 Text feature [results] present in test data point [True]
86 Text feature [different] present in test data point [True]
87 Text feature [pathway] present in test data point [True]
88 Text feature [figure] present in test data point [True]
90 Text feature [pathways] present in test data point [True]
91 Text feature [reported] present in test data point [True]
92 Text feature [studies] present in test data point [True]
```

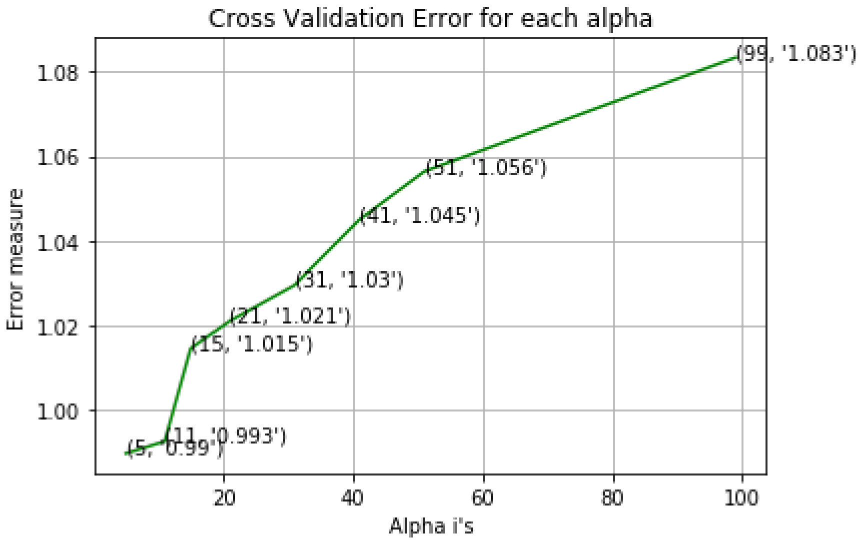
```
93 Text feature [observed] present in test data point [True]
94 Text feature [two] present in test data point [True]
95 Text feature [drug] present in test data point [True]
96 Text feature [proliferation] present in test data point [True]
97 Text feature [described] present in test data point [True]
98 Text feature [could] present in test data point [True]
99 Text feature [occur] present in test data point [True]
Out of the top  100  features  68 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 : 0.9897381065356475
for alpha = 11
Log Loss : 0.9925846381333129
for alpha = 15
Log Loss : 1.0145548226098013
for alpha = 21
Log Loss : 1.0211274311373892
for alpha = 31
Log Loss : 1.029602973240631
for alpha = 41
Log Loss : 1.0450068422912024
for alpha = 51
Log Loss : 1.0564016214688174
for alpha = 99
Log Loss : 1.0834159187843944
```



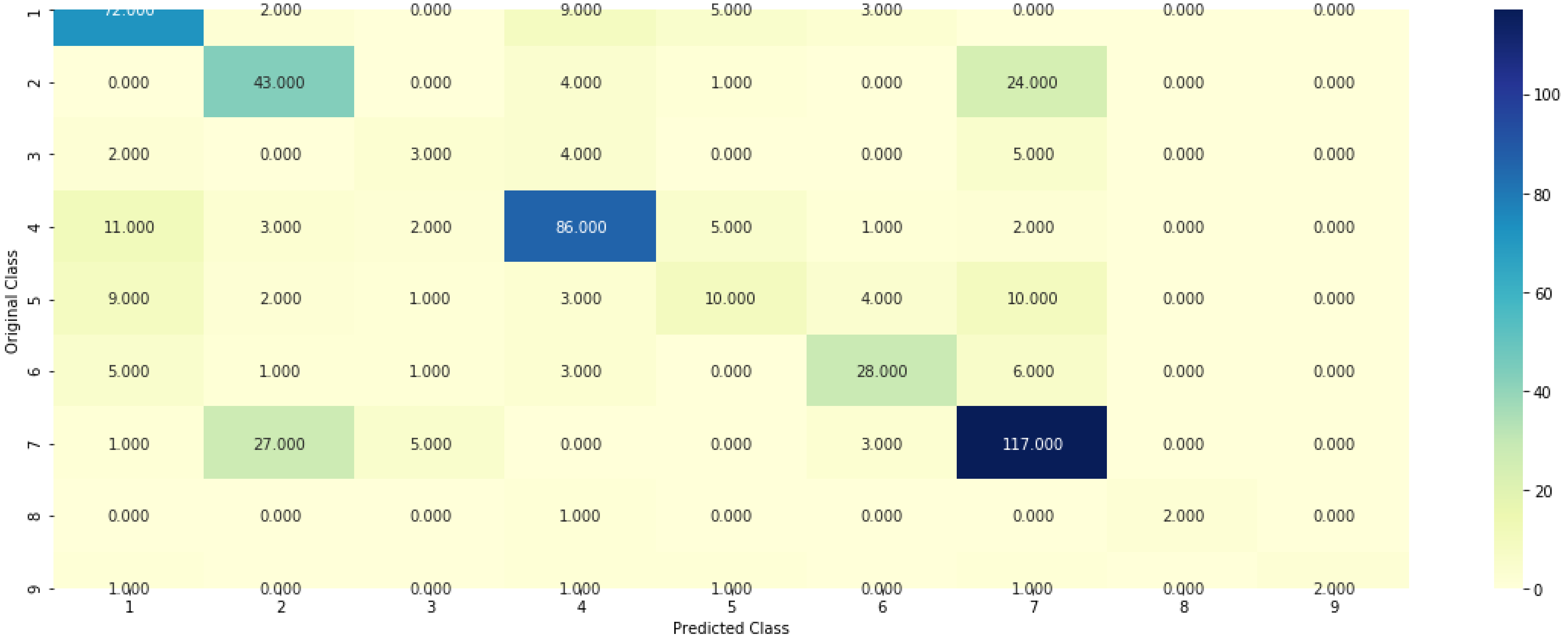
```
For values of best alpha =  5 The train log loss is: 0.4834016921784153
For values of best alpha =  5 The cross validation log loss is: 0.9897381065356475
For values of best alpha =  5 The test log loss is: 1.1149946934972625
```

4.2.2. Testing the model with best hyper paramters

In [0]:

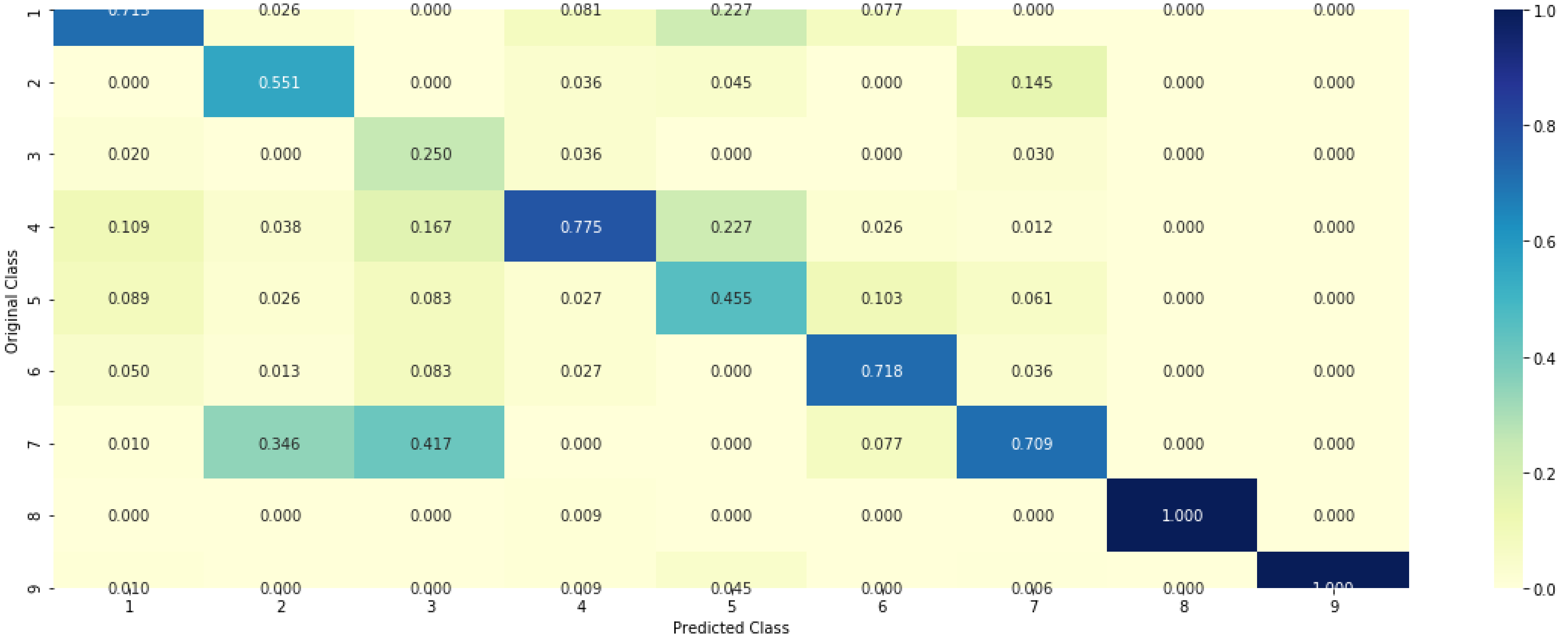
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 0.98973810653564475
Number of mis-classified points : 0.3176691729323308
-------------------- Confusion matrix --------------------
```



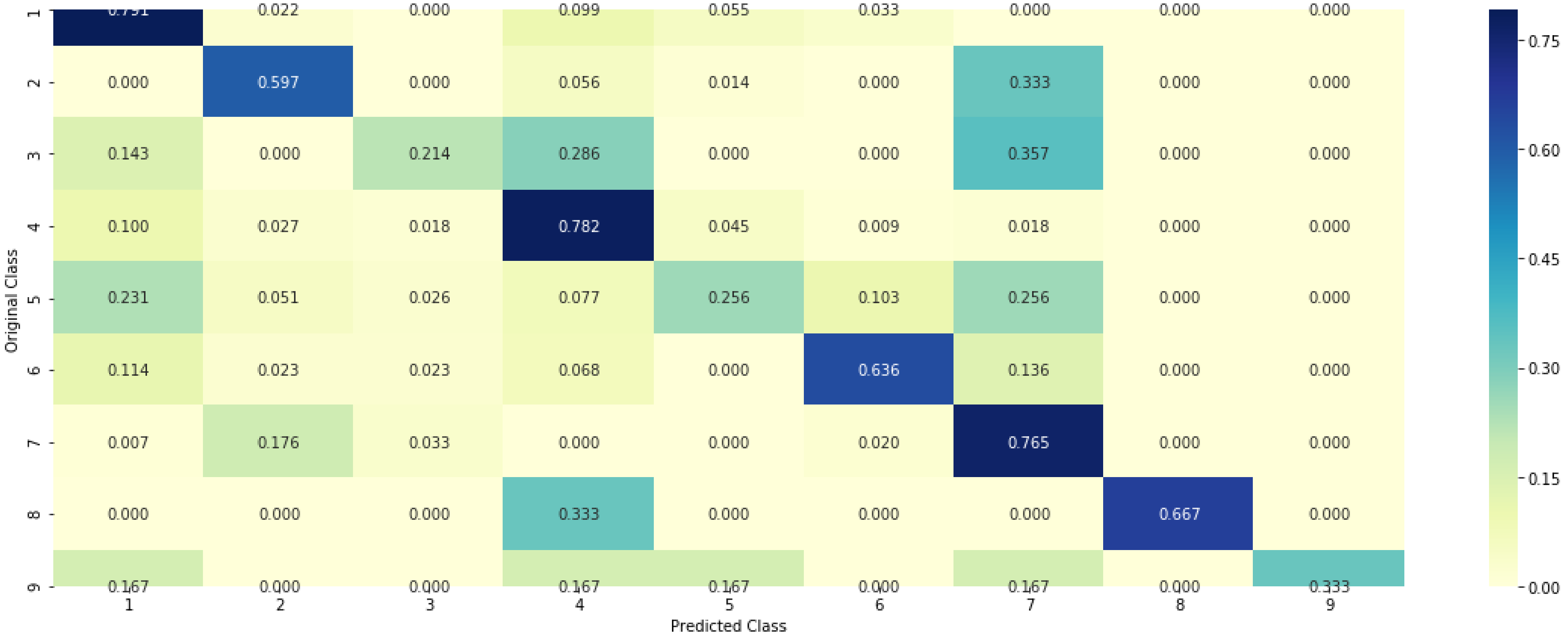
	1	2	3	4	5	6	7	8	9
1	72.000	2.000	0.000	9.000	5.000	3.000	0.000	0.000	0.000
2	0.000	43.000	0.000	4.000	1.000	0.000	24.000	0.000	0.000
3	2.000	0.000	3.000	4.000	0.000	0.000	5.000	0.000	0.000
4	11.000	3.000	2.000	86.000	5.000	1.000	2.000	0.000	0.000
5	9.000	2.000	1.000	3.000	10.000	4.000	10.000	0.000	0.000
6	5.000	1.000	1.000	3.000	0.000	28.000	6.000	0.000	0.000
7	1.000	27.000	5.000	0.000	0.000	3.000	117.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	0.000	2.000	0.000
9	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.713	0.026	0.000	0.081	0.227	0.077	0.000	0.000	0.000
2	0.000	0.551	0.000	0.036	0.045	0.000	0.145	0.000	0.000
3	0.020	0.000	0.250	0.036	0.000	0.000	0.030	0.000	0.000
4	0.109	0.038	0.167	0.775	0.227	0.026	0.012	0.000	0.000
5	0.089	0.026	0.083	0.027	0.455	0.103	0.061	0.000	0.000
6	0.050	0.013	0.083	0.027	0.000	0.718	0.036	0.000	0.000
7	0.010	0.346	0.417	0.000	0.000	0.077	0.709	0.000	0.000
8	0.000	0.000	0.000	0.009	0.000	0.000	0.000	1.000	0.000
9	0.010	0.000	0.000	0.009	0.045	0.000	0.006	0.000	1.000

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



In [14]:
```
# This improved slightly - 0.12
table.add_row(['All feature One-Hot','kNN', 0.98, 0.12])
print(table)
```

```
+---------------------+------------+---------------+------------+
|      Vectorizer     |   Model    | Test Log Loss | Improvement |
+---------------------+------------+---------------+------------+
|         NAN         |   Random   |      2.48     |      0     |
| Gene Feature One-Hot | Linear SVM |      1.19     |    0.01    |
| Var Feature One-Hot | Linear SVM |      1.73     |    0.0     |
| Text Feature One-Hot | Linear SVM |      1.1      |    0.09    |
|  TfIdf All Features | Naive Bayes |      1.17     |    0.1     |
| All feature One-Hot |    kNN     |      0.98     |    0.12    |
+---------------------+------------+---------------+------------+
```

4.2.3.Sample Query point -1

In [0]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 5
Actual Class : 4
The  5  nearest neighbours of the test points belongs to classes [4 4 6 4 4]
Fequency of nearest points : Counter({4: 4, 6: 1})
```

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("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

4.3. Logistic Regression

4.3.1. With Class balancing

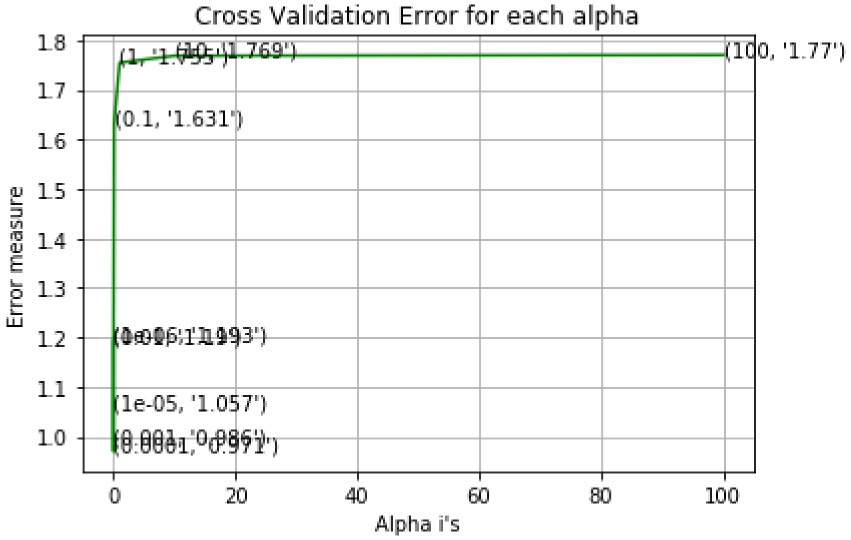
4.3.1.1. Hyper paramter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# 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.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 : 1.1927863023508278
for alpha = 1e-05
Log Loss : 1.0573691189802954
for alpha = 0.0001
Log Loss : 0.9705743175038444
for alpha = 0.001
Log Loss : 0.9863042435031497
for alpha = 0.01
Log Loss : 1.1895235795915284
for alpha = 0.1
Log Loss : 1.6312652057586234
for alpha = 1
Log Loss : 1.7547218052799338
for alpha = 10
Log Loss : 1.7686808799214384
for alpha = 100
Log Loss : 1.7702327025166267
```



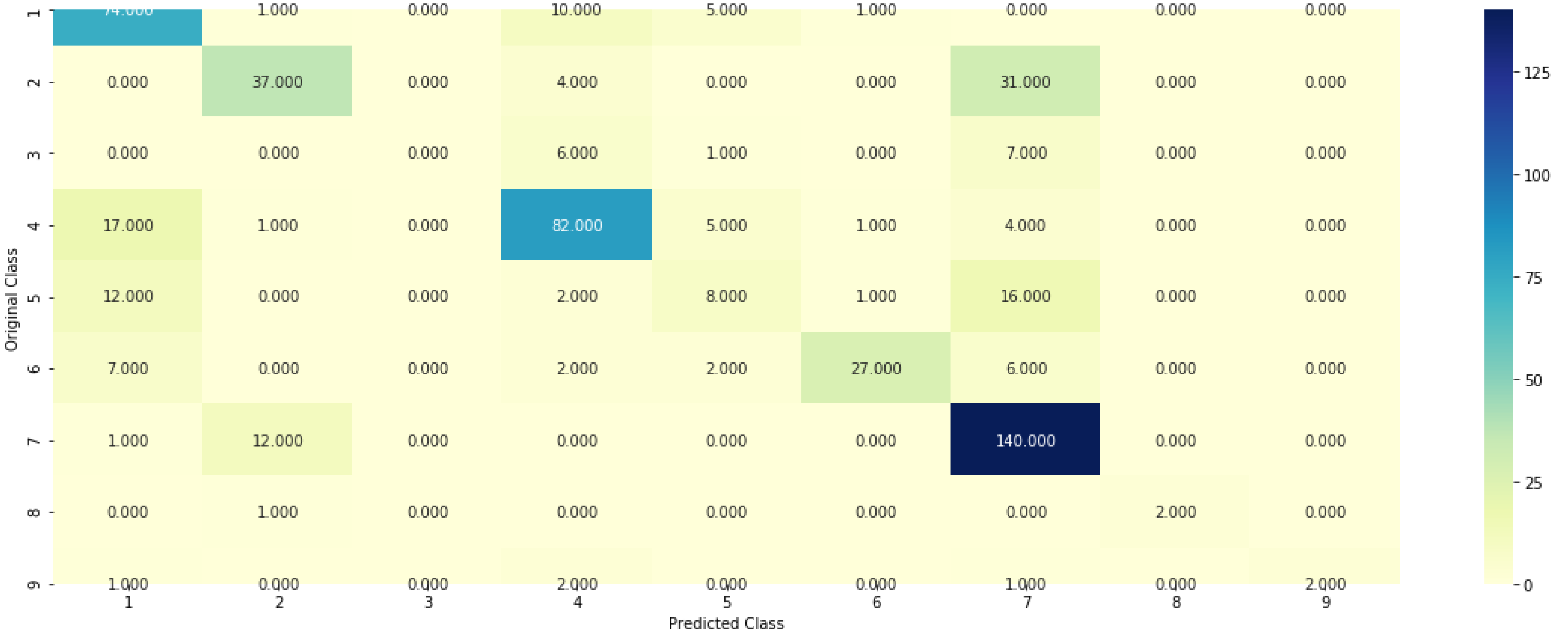
```
For values of best alpha =  0.0001 The train log loss is: 0.5845969374285752
For values of best alpha =  0.0001 The cross validation log loss is: 0.9705743175038444
For values of best alpha =  0.0001 The test log loss is: 1.0160381148200763
```

4.3.1.2. Testing the model with best hyper paramters

In [0]:

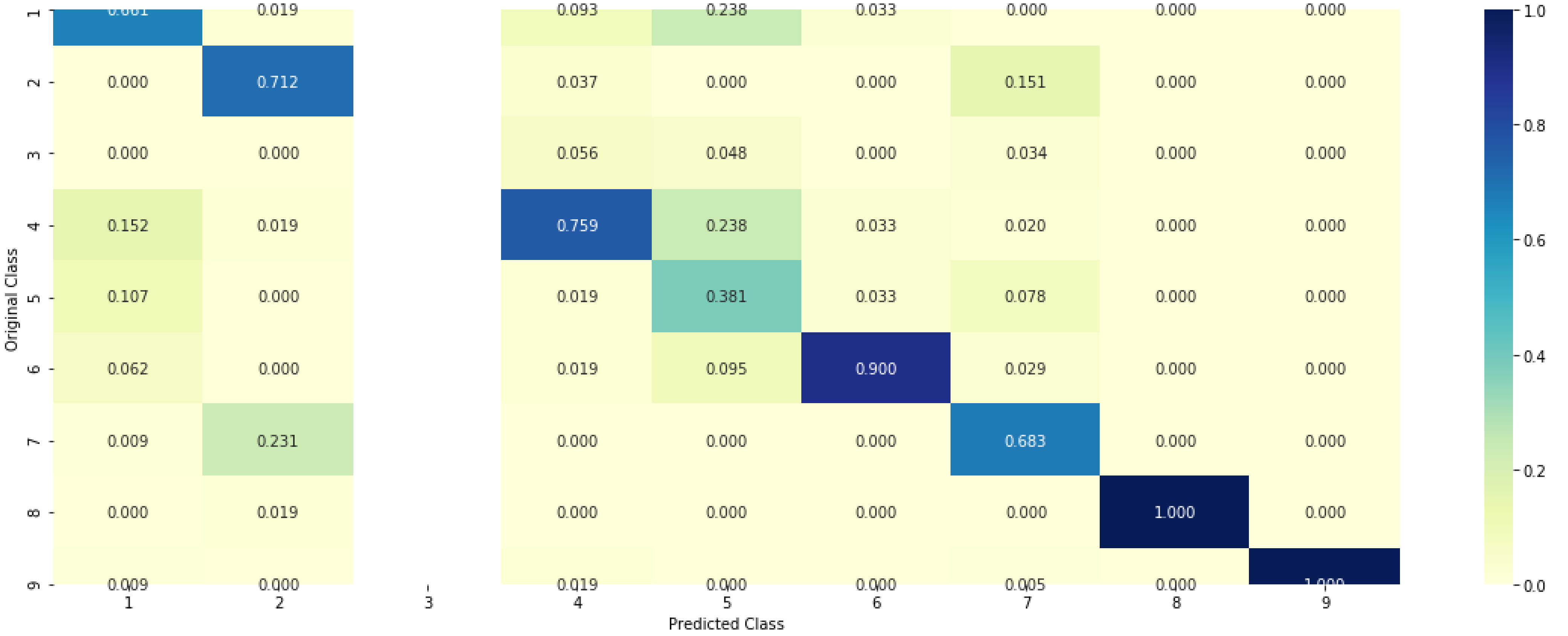
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

```
Log loss : 0.97057431175038444
Number of mis-classified points : 0.3007518796992481
-------------------- Confusion matrix --------------------
```



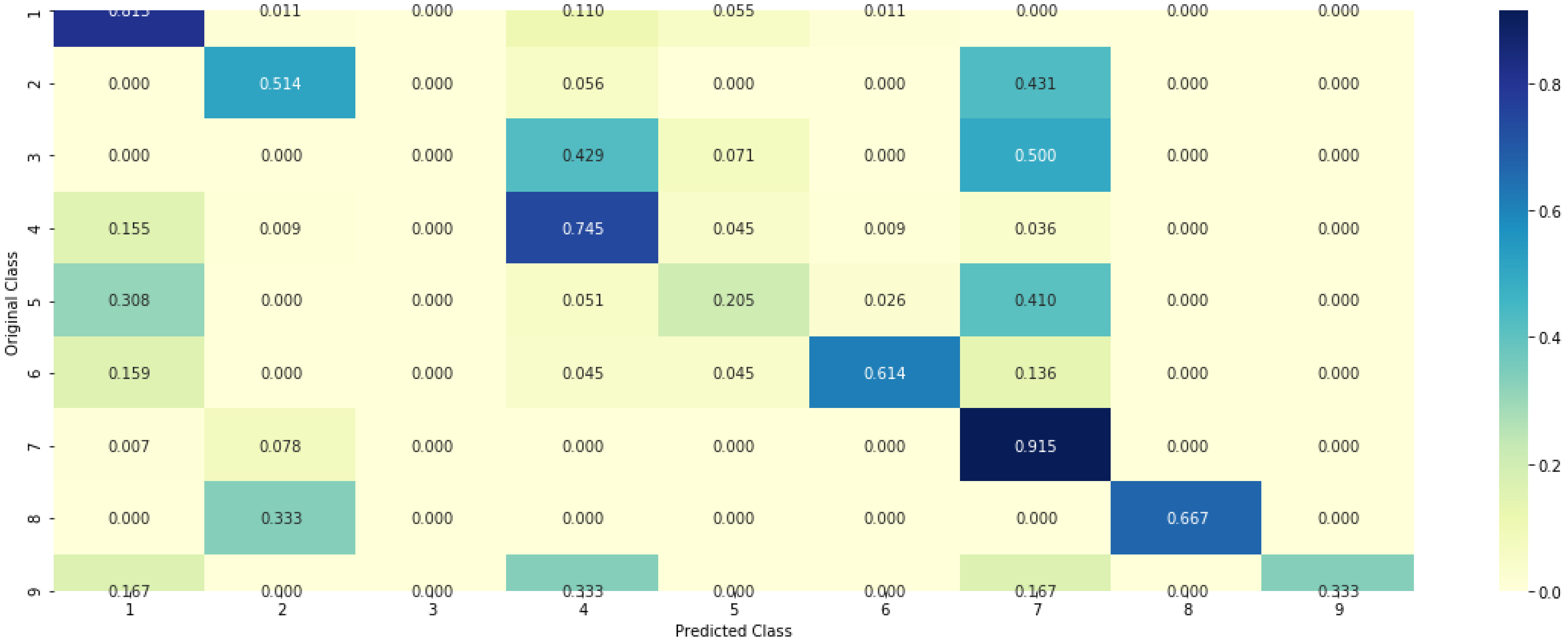
Original Class	1	2	3	4	5	6	7	8	9
1	74.000	1.000	0.000	10.000	5.000	1.000	0.000	0.000	0.000
2	0.000	37.000	0.000	4.000	0.000	0.000	31.000	0.000	0.000
3	0.000	0.000	0.000	6.000	1.000	0.000	7.000	0.000	0.000
4	17.000	1.000	0.000	82.000	5.000	1.000	4.000	0.000	0.000
5	12.000	0.000	0.000	2.000	8.000	1.000	16.000	0.000	0.000
6	7.000	0.000	0.000	2.000	2.000	27.000	6.000	0.000	0.000
7	1.000	12.000	0.000	0.000	0.000	0.000	140.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000
9	1.000	0.000	0.000	2.000	0.000	0.000	1.000	0.000	2.000

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



Original Class	1	2	3	4	5	6	7	8	9
1	0.661	0.019		0.093	0.238	0.033	0.000	0.000	0.000
2	0.000	0.712		0.037	0.000	0.000	0.151	0.000	0.000
3	0.000	0.000		0.056	0.048	0.000	0.034	0.000	0.000
4	0.152	0.019		0.759	0.238	0.033	0.020	0.000	0.000
5	0.107	0.000		0.019	0.381	0.033	0.078	0.000	0.000
6	0.062	0.000		0.019	0.095	0.900	0.029	0.000	0.000
7	0.009	0.231		0.000	0.000	0.000	0.683	0.000	0.000
8	0.000	0.019		0.000	0.000	0.000	0.000	1.000	0.000
9	0.009	0.000		0.019	0.000	0.000	0.005	0.000	1.000

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



In [16]:

```
# This is a huge improvement - 0.24
table.add_row(['All feature One-Hot','Logistic', 0.97, 0.24])
print(table)
```

```
+---------------------+-------------+--------------+-------------+
|      Vectorizer     |    Model    | Test Log Loss | Improvement |
+---------------------+-------------+--------------+-------------+
|         NAN         |    Random   |     2.48     |      0      |
| Gene Feature One-Hot |  Linear SVM |     1.19     |     0.01    |
| Var Feature One-Hot  |  Linear SVM |     1.73     |     0.0     |
| Text Feature One-Hot |  Linear SVM |     1.1      |     0.09    |
|  TfIdf All Features  | Naive Bayes |     1.17     |     0.1     |
| All feature One-Hot  |     kNN     |     0.98     |     0.12    |
| All feature One-Hot  |   Logistic  |     0.97     |     0.24    |
+---------------------+-------------+--------------+-------------+
```

4.3.1.3. Feature Importance

In [0]:

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

4.3.1.3.1. Correctly Classified point

In [0]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0132 0.0119 0.0055 0.8752 0.012  0.0572 0.0195 0.0032 0.0023]]
Actual Class : 4
--------------------------------------------------
36 Text feature [missense] present in test data point [True]
45 Text feature [families] present in test data point [True]
60 Text feature [suppressor] present in test data point [True]
114 Text feature [suggesting] present in test data point [True]
148 Text feature [germline] present in test data point [True]
158 Text feature [protein] present in test data point [True]
166 Text feature [low] present in test data point [True]
172 Text feature [dna] present in test data point [True]
175 Text feature [ref] present in test data point [True]
212 Text feature [family] present in test data point [True]
221 Text feature [members] present in test data point [True]
252 Text feature [catalytic] present in test data point [True]
265 Text feature [suggested] present in test data point [True]
278 Text feature [lines] present in test data point [True]
289 Text feature [kinases] present in test data point [True]
293 Text feature [contribute] present in test data point [True]
296 Text feature [cycle] present in test data point [True]
325 Text feature [risk] present in test data point [True]
326 Text feature [mutants] present in test data point [True]
345 Text feature [impaired] present in test data point [True]
366 Text feature [primary] present in test data point [True]
401 Text feature [proteins] present in test data point [True]
408 Text feature [genetic] present in test data point [True]
410 Text feature [recent] present in test data point [True]
417 Text feature [deletion] present in test data point [True]
420 Text feature [negative] present in test data point [True]
433 Text feature [described] present in test data point [True]
444 Text feature [activity] present in test data point [True]
482 Text feature [consistent] present in test data point [True]
Out of the top  500  features  29 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [0]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[9.500e-03 7.500e-03 4.200e-03 3.730e-02 7.000e-03 1.000e-03 9.324e-01
  6.000e-04 5.000e-04]]
Actual Class : 7
--------------------------------------------------
6 Text feature [enhanced] present in test data point [True]
16 Text feature [activation] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
19 Text feature [constitutive] present in test data point [True]
31 Text feature [activated] present in test data point [True]
32 Text feature [transformed] present in test data point [True]
37 Text feature [presence] present in test data point [True]
41 Text feature [3b] present in test data point [True]
67 Text feature [expressing] present in test data point [True]
77 Text feature [2a] present in test data point [True]
80 Text feature [total] present in test data point [True]
82 Text feature [constitutively] present in test data point [True]
83 Text feature [increased] present in test data point [True]
85 Text feature [activate] present in test data point [True]
88 Text feature [occur] present in test data point [True]
89 Text feature [long] present in test data point [True]
92 Text feature [days] present in test data point [True]
96 Text feature [pathways] present in test data point [True]
107 Text feature [phospho] present in test data point [True]
108 Text feature [transforming] present in test data point [True]
120 Text feature [signaling] present in test data point [True]
135 Text feature [factor] present in test data point [True]
140 Text feature [concentrations] present in test data point [True]
149 Text feature [24] present in test data point [True]
155 Text feature [activating] present in test data point [True]
163 Text feature [overexpression] present in test data point [True]
170 Text feature [blot] present in test data point [True]
182 Text feature [reverse] present in test data point [True]
200 Text feature [genomic] present in test data point [True]
209 Text feature [25] present in test data point [True]
210 Text feature [membrane] present in test data point [True]
212 Text feature [absence] present in test data point [True]
216 Text feature [medium] present in test data point [True]
224 Text feature [sensitive] present in test data point [True]
231 Text feature [inhibitor] present in test data point [True]
255 Text feature [positive] present in test data point [True]
260 Text feature [erk] present in test data point [True]
272 Text feature [additional] present in test data point [True]
273 Text feature [bp] present in test data point [True]
274 Text feature [transformation] present in test data point [True]
282 Text feature [gfp] present in test data point [True]
283 Text feature [regulated] present in test data point [True]
284 Text feature [common] present in test data point [True]
286 Text feature [sensitivity] present in test data point [True]
290 Text feature [2b] present in test data point [True]
314 Text feature [drug] present in test data point [True]
316 Text feature [provided] present in test data point [True]
319 Text feature [3a] present in test data point [True]
331 Text feature [download] present in test data point [True]
344 Text feature [mutant] present in test data point [True]
348 Text feature [oncogenic] present in test data point [True]
357 Text feature [inhibition] present in test data point [True]
360 Text feature [transduced] present in test data point [True]
365 Text feature [factors] present in test data point [True]
367 Text feature [strongly] present in test data point [True]
369 Text feature [14] present in test data point [True]
372 Text feature [4a] present in test data point [True]
374 Text feature [years] present in test data point [True]
376 Text feature [previously] present in test data point [True]
380 Text feature [cells] present in test data point [True]
```

```
383 Text feature [hr] present in test data point [True]
388 Text feature [hours] present in test data point [True]
392 Text feature [events] present in test data point [True]
394 Text feature [survival] present in test data point [True]
398 Text feature [phosphorylation] present in test data point [True]
400 Text feature [without] present in test data point [True]
403 Text feature [tumor] present in test data point [True]
412 Text feature [dose] present in test data point [True]
416 Text feature [subsequent] present in test data point [True]
418 Text feature [stat3] present in test data point [True]
426 Text feature [use] present in test data point [True]
436 Text feature [furthermore] present in test data point [True]
439 Text feature [found] present in test data point [True]
444 Text feature [indicate] present in test data point [True]
445 Text feature [four] present in test data point [True]
446 Text feature [tyrosine] present in test data point [True]
452 Text feature [vector] present in test data point [True]
453 Text feature [leukemia] present in test data point [True]
454 Text feature [properties] present in test data point [True]
458 Text feature [gain] present in test data point [True]
467 Text feature [express] present in test data point [True]
481 Text feature [signals] present in test data point [True]
484 Text feature [observations] present in test data point [True]
485 Text feature [within] present in test data point [True]
496 Text feature [plasmid] present in test data point [True]
Out of the top  500  features  85 are present in query point
```

4.3.2. Without Class balancing

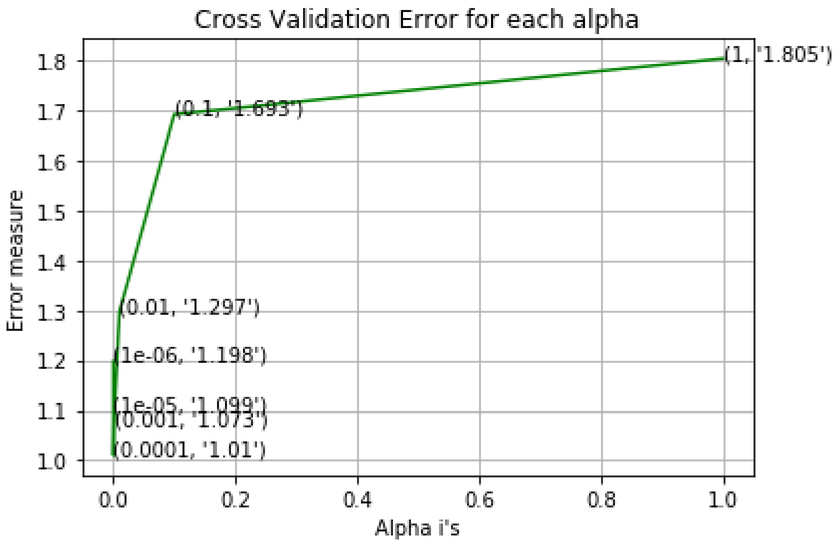
4.3.2.1. Hyper paramter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.19799407595299
for alpha = 1e-05
Log Loss : 1.0985309539834947
for alpha = 0.0001
Log Loss : 1.0100689088426227
for alpha = 0.001
Log Loss : 1.0727835429612813
for alpha = 0.01
Log Loss : 1.2970402186376642
for alpha = 0.1
Log Loss : 1.6925015250316762
for alpha = 1
Log Loss : 1.8045119585096452
```



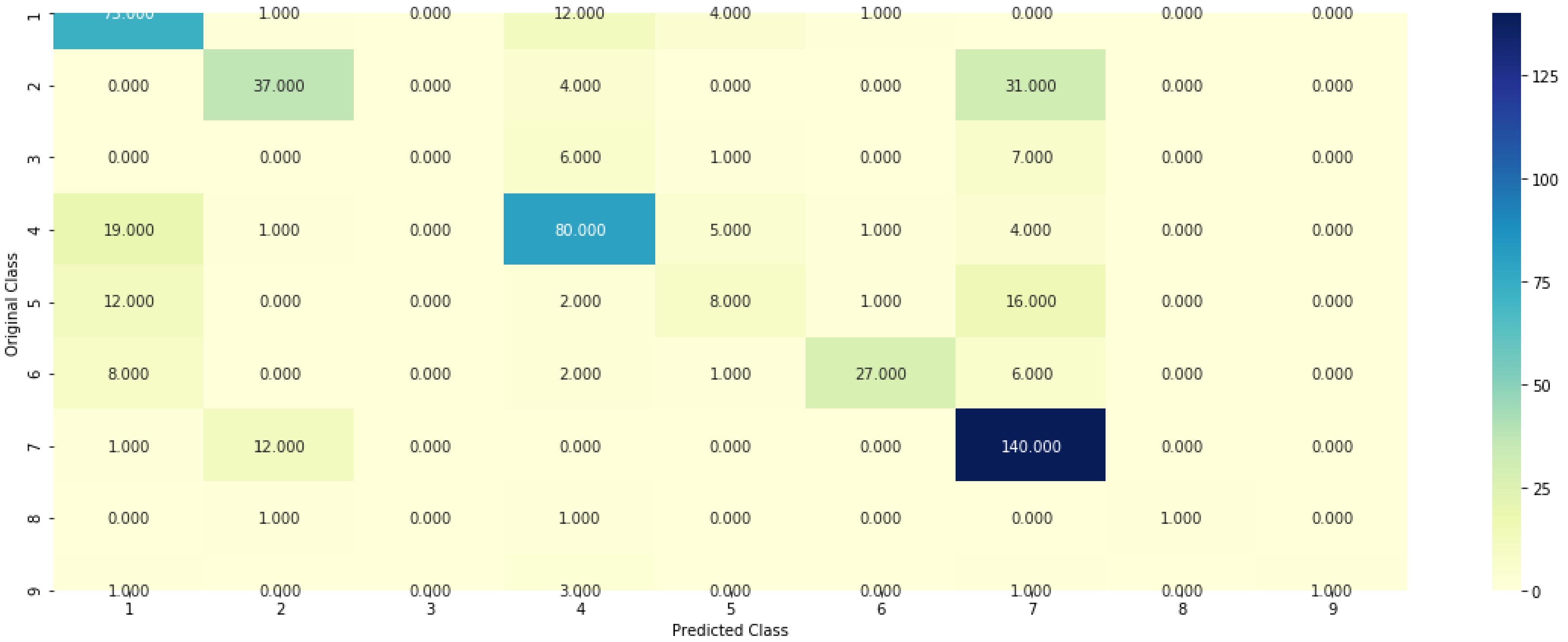
```
For values of best alpha =  0.0001 The train log loss is: 0.5698664562221166
For values of best alpha =  0.0001 The cross validation log loss is: 1.0100689088426227
For values of best alpha =  0.0001 The test log loss is: 1.0328095681457892
```

4.3.2.2. Testing model with best hyper parameters

In [0]:

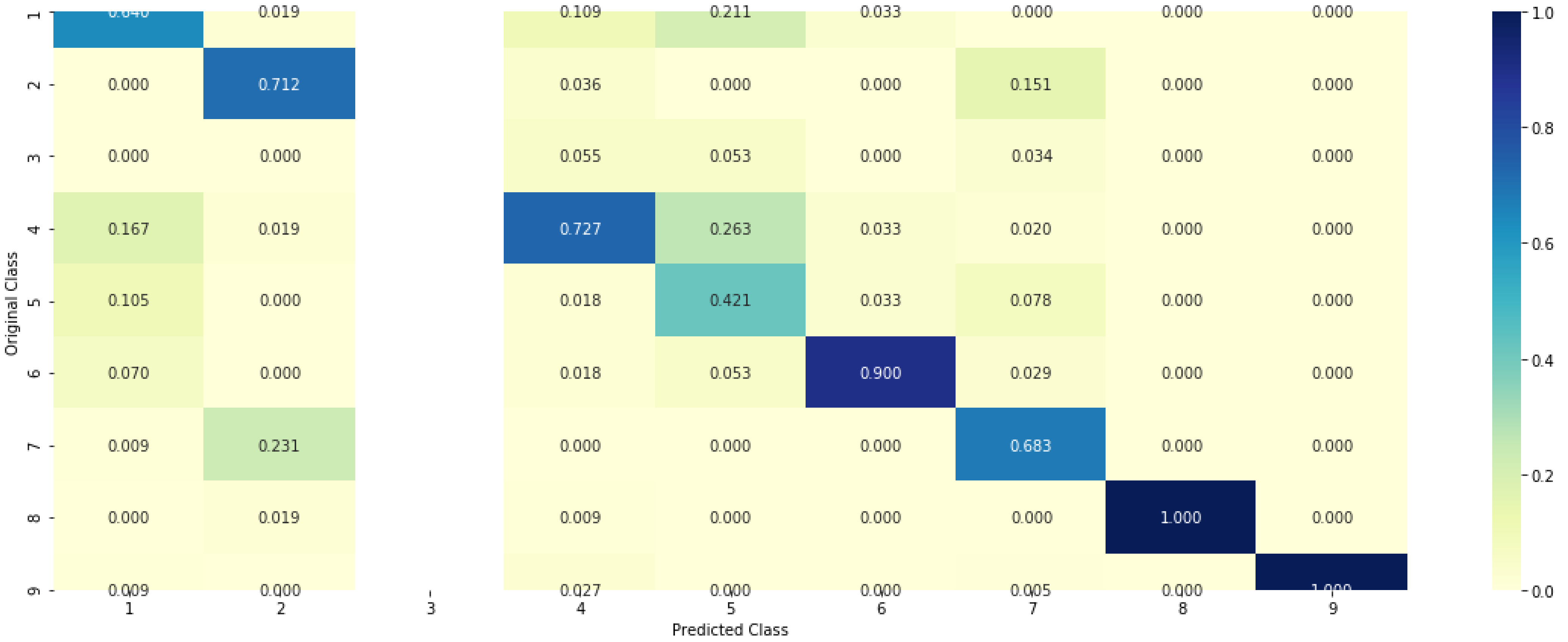
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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 : 1.0100689088426227
Number of mis-classified points : 0.3101503759398496
-------------------- Confusion matrix --------------------
```



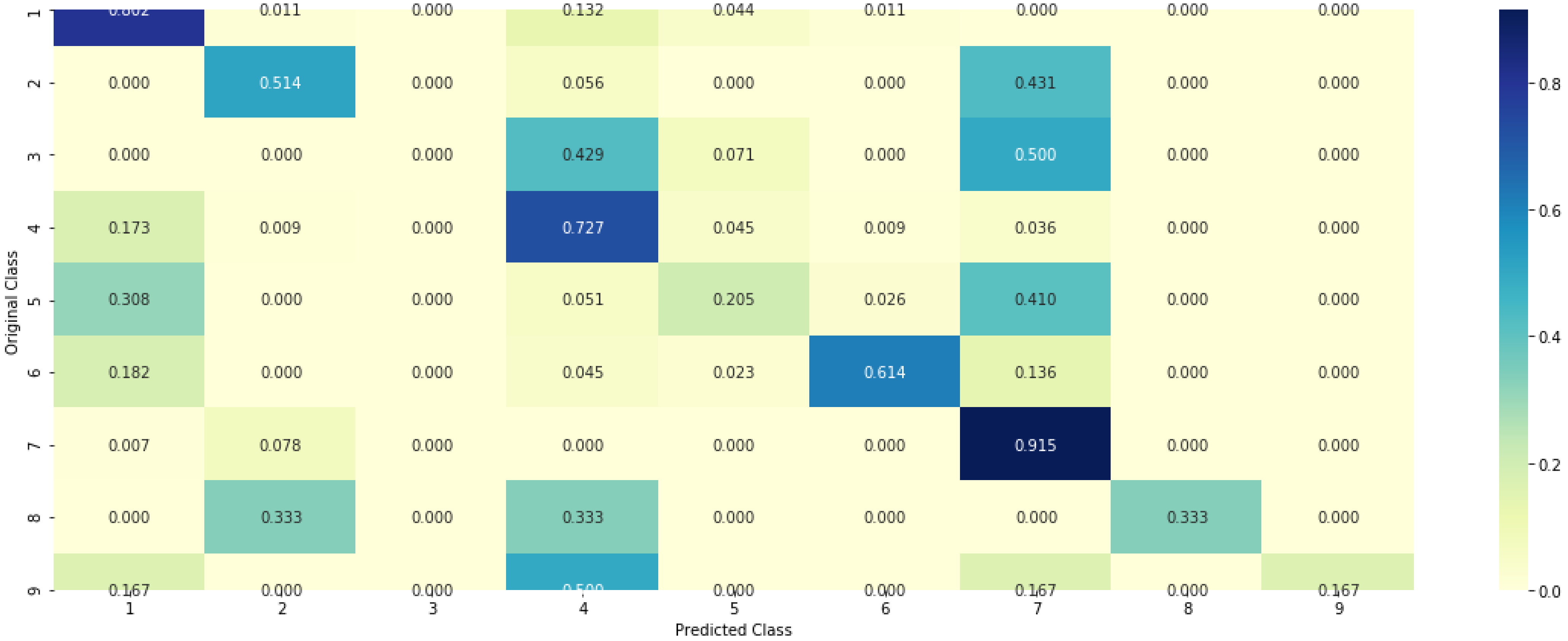
	1	2	3	4	5	6	7	8	9
1	73.000	1.000	0.000	12.000	4.000	1.000	0.000	0.000	0.000
2	0.000	37.000	0.000	4.000	0.000	0.000	31.000	0.000	0.000
3	0.000	0.000	0.000	6.000	1.000	0.000	7.000	0.000	0.000
4	19.000	1.000	0.000	80.000	5.000	1.000	4.000	0.000	0.000
5	12.000	0.000	0.000	2.000	8.000	1.000	16.000	0.000	0.000
6	8.000	0.000	0.000	2.000	1.000	27.000	6.000	0.000	0.000
7	1.000	12.000	0.000	0.000	0.000	0.000	140.000	0.000	0.000
8	0.000	1.000	0.000	1.000	0.000	0.000	0.000	1.000	0.000
9	1.000	0.000	0.000	3.000	0.000	0.000	1.000	0.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.640	0.019		0.109	0.211	0.033	0.000	0.000	0.000
2	0.000	0.712		0.036	0.000	0.000	0.151	0.000	0.000
3	0.000	0.000		0.055	0.053	0.000	0.034	0.000	0.000
4	0.167	0.019		0.727	0.263	0.033	0.020	0.000	0.000
5	0.105	0.000		0.018	0.421	0.033	0.078	0.000	0.000
6	0.070	0.000		0.018	0.053	0.900	0.029	0.000	0.000
7	0.009	0.231		0.000	0.000	0.000	0.683	0.000	0.000
8	0.000	0.019		0.009	0.000	0.000	0.000	1.000	0.000
9	0.009	0.000		0.027	0.000	0.000	0.005	0.000	1.000

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



In [18]:

```
# This is a good improvement - 0.24
table.add_row(['All feature One-Hot','Logistic Reg', 1.01, 0.24])
print(table)
```

```
+---------------------+--------------+---------------+-------------+
|      Vectorizer     |    Model     | Test Log Loss | Improvement |
+---------------------+--------------+---------------+-------------+
|         NAN         |    Random    |      2.48     |      0      |
| Gene Feature One-Hot |  Linear SVM  |      1.19     |     0.01    |
| Var Feature One-Hot |  Linear SVM  |      1.73     |     0.0     |
| Text Feature One-Hot |  Linear SVM  |      1.1      |     0.09    |
|  TfIdf All Features | Naive Bayes  |      1.17     |     0.1     |
| All feature One-Hot |     kNN      |      0.98     |     0.12    |
| All feature One-Hot |   Logistic   |      0.97     |     0.24    |
| All feature One-Hot | Logistic Reg |      1.01     |     0.24    |
+---------------------+--------------+---------------+-------------+
```

4.3.2.3. Feature Importance, Correctly Classified point

In [0]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0132 0.012  0.0046 0.8883 0.0106 0.048  0.0205 0.0015 0.0013]]
Actual Class : 4
--------------------------------------------------
41 Text feature [missense] present in test data point [True]
60 Text feature [families] present in test data point [True]
70 Text feature [suppressor] present in test data point [True]
107 Text feature [suggesting] present in test data point [True]
125 Text feature [protein] present in test data point [True]
140 Text feature [low] present in test data point [True]
149 Text feature [ref] present in test data point [True]
154 Text feature [germline] present in test data point [True]
162 Text feature [dna] present in test data point [True]
198 Text feature [suggested] present in test data point [True]
222 Text feature [lines] present in test data point [True]
226 Text feature [family] present in test data point [True]
231 Text feature [members] present in test data point [True]
239 Text feature [catalytic] present in test data point [True]
260 Text feature [kinases] present in test data point [True]
281 Text feature [contribute] present in test data point [True]
294 Text feature [impaired] present in test data point [True]
312 Text feature [mutants] present in test data point [True]
314 Text feature [cycle] present in test data point [True]
340 Text feature [risk] present in test data point [True]
378 Text feature [primary] present in test data point [True]
382 Text feature [recent] present in test data point [True]
389 Text feature [proteins] present in test data point [True]
399 Text feature [described] present in test data point [True]
406 Text feature [genetic] present in test data point [True]
412 Text feature [deletion] present in test data point [True]
447 Text feature [activity] present in test data point [True]
460 Text feature [consistent] present in test data point [True]
461 Text feature [negative] present in test data point [True]
486 Text feature [whereas] present in test data point [True]
494 Text feature [previously] present in test data point [True]
Out of the top  500  features  31 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[9.200e-03 7.500e-03 2.600e-03 5.120e-02 6.300e-03 8.000e-04 9.221e-01
  2.000e-04 1.000e-04]]
Actual Class : 7
--------------------------------------------------
5 Text feature [enhanced] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
27 Text feature [activation] present in test data point [True]
34 Text feature [3b] present in test data point [True]
36 Text feature [presence] present in test data point [True]
37 Text feature [activated] present in test data point [True]
46 Text feature [transformed] present in test data point [True]
74 Text feature [increased] present in test data point [True]
88 Text feature [occur] present in test data point [True]
89 Text feature [expressing] present in test data point [True]
94 Text feature [2a] present in test data point [True]
96 Text feature [days] present in test data point [True]
101 Text feature [total] present in test data point [True]
120 Text feature [activate] present in test data point [True]
125 Text feature [blot] present in test data point [True]
132 Text feature [24] present in test data point [True]
137 Text feature [long] present in test data point [True]
138 Text feature [concentrations] present in test data point [True]
140 Text feature [phospho] present in test data point [True]
142 Text feature [factor] present in test data point [True]
143 Text feature [pathways] present in test data point [True]
144 Text feature [overexpression] present in test data point [True]
146 Text feature [constitutively] present in test data point [True]
155 Text feature [genomic] present in test data point [True]
161 Text feature [activating] present in test data point [True]
164 Text feature [reverse] present in test data point [True]
168 Text feature [signaling] present in test data point [True]
176 Text feature [membrane] present in test data point [True]
179 Text feature [additional] present in test data point [True]
184 Text feature [25] present in test data point [True]
192 Text feature [common] present in test data point [True]
207 Text feature [4a] present in test data point [True]
208 Text feature [transforming] present in test data point [True]
211 Text feature [positive] present in test data point [True]
212 Text feature [sensitivity] present in test data point [True]
220 Text feature [bp] present in test data point [True]
222 Text feature [regulated] present in test data point [True]
224 Text feature [medium] present in test data point [True]
244 Text feature [sensitive] present in test data point [True]
245 Text feature [gfp] present in test data point [True]
251 Text feature [dose] present in test data point [True]
252 Text feature [inhibitor] present in test data point [True]
256 Text feature [provided] present in test data point [True]
259 Text feature [mutant] present in test data point [True]
266 Text feature [2b] present in test data point [True]
279 Text feature [3a] present in test data point [True]
281 Text feature [erk] present in test data point [True]
288 Text feature [hr] present in test data point [True]
293 Text feature [absence] present in test data point [True]
304 Text feature [download] present in test data point [True]
318 Text feature [subsequent] present in test data point [True]
319 Text feature [previously] present in test data point [True]
323 Text feature [four] present in test data point [True]
326 Text feature [factors] present in test data point [True]
328 Text feature [drug] present in test data point [True]
330 Text feature [without] present in test data point [True]
345 Text feature [hours] present in test data point [True]
346 Text feature [strongly] present in test data point [True]
349 Text feature [indicate] present in test data point [True]
```

```
350 Text feature [transformation] present in test data point [True]
356 Text feature [events] present in test data point [True]
358 Text feature [cells] present in test data point [True]
360 Text feature [years] present in test data point [True]
362 Text feature [tumor] present in test data point [True]
384 Text feature [transduced] present in test data point [True]
386 Text feature [gain] present in test data point [True]
387 Text feature [use] present in test data point [True]
390 Text feature [survival] present in test data point [True]
401 Text feature [furthermore] present in test data point [True]
405 Text feature [stat3] present in test data point [True]
415 Text feature [properties] present in test data point [True]
419 Text feature [14] present in test data point [True]
420 Text feature [vector] present in test data point [True]
424 Text feature [within] present in test data point [True]
426 Text feature [phosphorylation] present in test data point [True]
438 Text feature [primer] present in test data point [True]
439 Text feature [observations] present in test data point [True]
442 Text feature [leukemia] present in test data point [True]
445 Text feature [mutants] present in test data point [True]
447 Text feature [plasmid] present in test data point [True]
449 Text feature [oncogenic] present in test data point [True]
451 Text feature [found] present in test data point [True]
458 Text feature [inhibition] present in test data point [True]
460 Text feature [40] present in test data point [True]
466 Text feature [region] present in test data point [True]
468 Text feature [together] present in test data point [True]
470 Text feature [progression] present in test data point [True]
477 Text feature [recently] present in test data point [True]
478 Text feature [signals] present in test data point [True]
479 Text feature [assessed] present in test data point [True]
480 Text feature [20] present in test data point [True]
487 Text feature [express] present in test data point [True]
490 Text feature [green] present in test data point [True]
491 Text feature [respectively] present in test data point [True]
492 Text feature [transfection] present in test data point [True]
494 Text feature [current] present in test data point [True]
496 Text feature [domain] present in test data point [True]
Out of the top  500  features  98 are present in query point
```

4.4. Linear Support Vector Machines

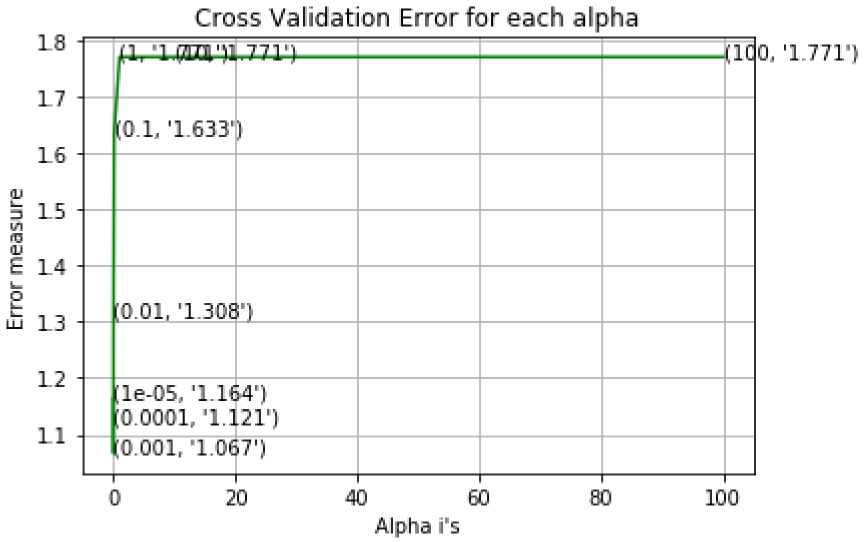
4.4.1. Hyper paramter tuning

In [0]:

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

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, 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 paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', 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 : 1.1638476100243025
for C = 0.0001
Log Loss : 1.1206314327185705
for C = 0.001
Log Loss : 1.0666949292072911
for C = 0.01
Log Loss : 1.30824325199411
for C = 0.1
Log Loss : 1.6327338951884551
for C = 1
Log Loss : 1.770548777840316
for C = 10
Log Loss : 1.770548001685657
for C = 100
Log Loss : 1.7705480340245017
```

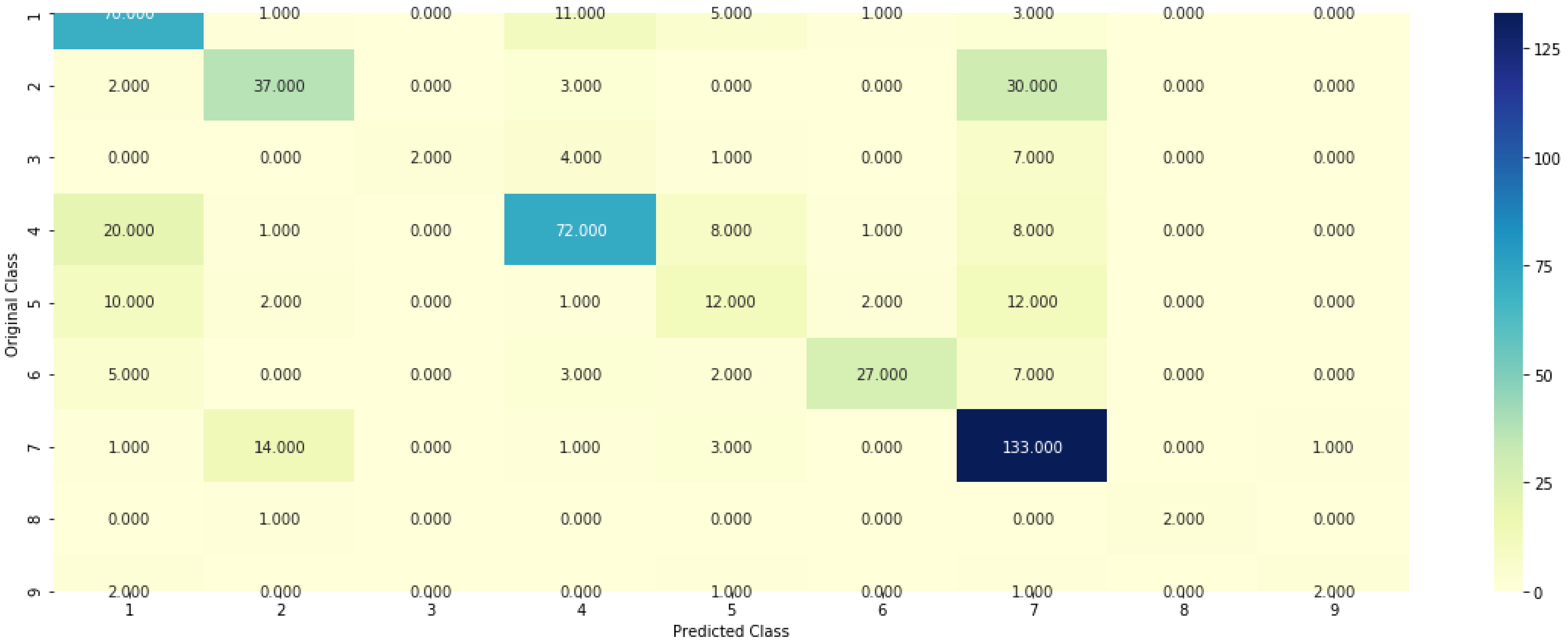


```
For values of best alpha =  0.001 The train log loss is: 0.8234321175044713
For values of best alpha =  0.001 The cross validation log loss is: 1.0666949292072911
For values of best alpha =  0.001 The test log loss is: 1.1464186790087711
```

4.4.2. Testing model with best hyper parameters

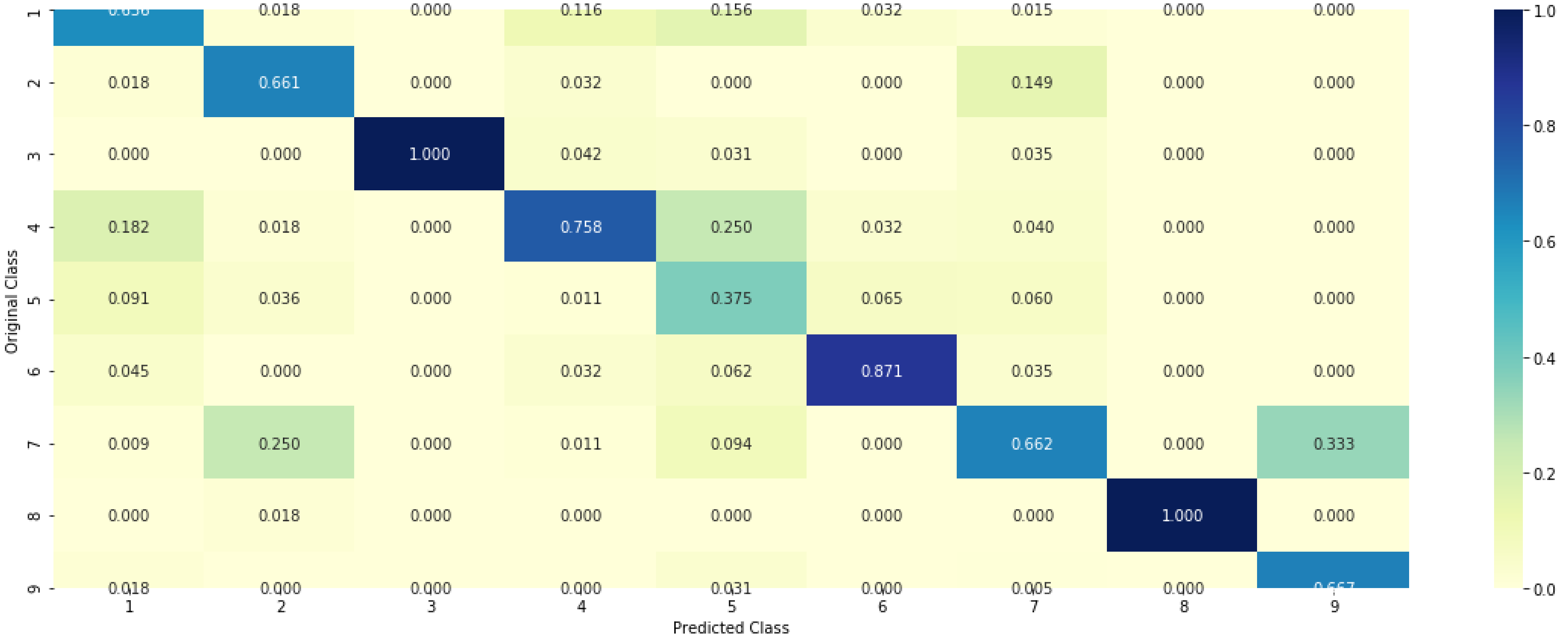
```
In [0]: # read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

        # --------------------------------
        # default parameters
        # SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, 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 : 1.0666949292072911
Number of mis-classified points : 0.32894736842105265
-------------------- Confusion matrix --------------------
```



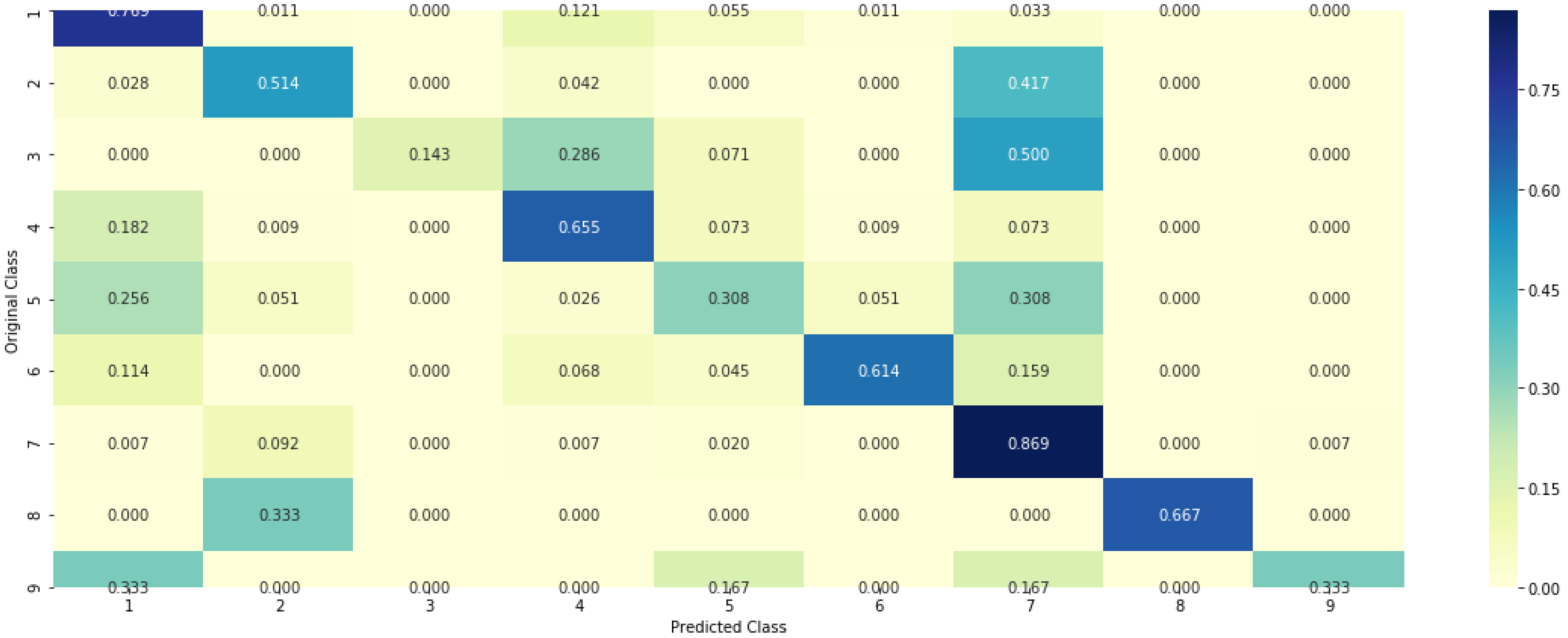
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	70.000	1.000	0.000	11.000	5.000	1.000	3.000	0.000	0.000
2	2.000	37.000	0.000	3.000	0.000	0.000	30.000	0.000	0.000
3	0.000	0.000	2.000	4.000	1.000	0.000	7.000	0.000	0.000
4	20.000	1.000	0.000	72.000	8.000	1.000	8.000	0.000	0.000
5	10.000	2.000	0.000	1.000	12.000	2.000	12.000	0.000	0.000
6	5.000	0.000	0.000	3.000	2.000	27.000	7.000	0.000	0.000
7	1.000	14.000	0.000	1.000	3.000	0.000	133.000	0.000	1.000
8	0.000	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000
9	2.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	2.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.636	0.018	0.000	0.116	0.156	0.032	0.015	0.000	0.000
2	0.018	0.661	0.000	0.032	0.000	0.000	0.149	0.000	0.000
3	0.000	0.000	1.000	0.042	0.031	0.000	0.035	0.000	0.000
4	0.182	0.018	0.000	0.758	0.250	0.032	0.040	0.000	0.000
5	0.091	0.036	0.000	0.011	0.375	0.065	0.060	0.000	0.000
6	0.045	0.000	0.000	0.032	0.062	0.871	0.035	0.000	0.000
7	0.009	0.250	0.000	0.011	0.094	0.000	0.662	0.000	0.333
8	0.000	0.018	0.000	0.000	0.000	0.000	0.000	1.000	0.000
9	0.018	0.000	0.000	0.000	0.031	0.000	0.005	0.000	0.667

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



In [20]:

```
# This improved a lot - 0.18
table.add_row(['All feature One-Hot','kNN', 1.06, 0.18])
print(table)
```

```
+---------------------+-------------+---------------+-------------+
|      Vectorizer     |    Model    | Test Log Loss | Improvement |
+---------------------+-------------+---------------+-------------+
|         NAN         |    Random   |      2.48     |      0      |
| Gene Feature One-Hot|  Linear SVM |      1.19     |     0.01    |
| Var Feature One-Hot |  Linear SVM |      1.73     |     0.0     |
| Text Feature One-Hot|  Linear SVM |      1.1      |     0.09    |
|  TfIdf All Features | Naive Bayes |      1.17     |     0.1     |
| All feature One-Hot |     kNN     |      0.98     |     0.12    |
| All feature One-Hot |   Logistic  |      0.97     |     0.24    |
| All feature One-Hot | Logistic Reg|      1.01     |     0.24    |
| All feature One-Hot |     kNN     |      1.06     |     0.18    |
+---------------------+-------------+---------------+-------------+
```

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [0]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0289 0.033  0.0087 0.7717 0.0232 0.0253 0.1026 0.0032 0.0034]]
Actual Class : 4
--------------------------------------------------
20 Text feature [families] present in test data point [True]
21 Text feature [suppressor] present in test data point [True]
22 Text feature [suggesting] present in test data point [True]
26 Text feature [missense] present in test data point [True]
168 Text feature [catalytic] present in test data point [True]
190 Text feature [germline] present in test data point [True]
216 Text feature [family] present in test data point [True]
224 Text feature [kinases] present in test data point [True]
240 Text feature [mutants] present in test data point [True]
253 Text feature [low] present in test data point [True]
290 Text feature [dna] present in test data point [True]
298 Text feature [lines] present in test data point [True]
306 Text feature [members] present in test data point [True]
311 Text feature [ref] present in test data point [True]
328 Text feature [protein] present in test data point [True]
334 Text feature [negative] present in test data point [True]
367 Text feature [contribute] present in test data point [True]
368 Text feature [cycle] present in test data point [True]
388 Text feature [recent] present in test data point [True]
407 Text feature [primary] present in test data point [True]
432 Text feature [previously] present in test data point [True]
445 Text feature [activity] present in test data point [True]
453 Text feature [described] present in test data point [True]
464 Text feature [consistent] present in test data point [True]
479 Text feature [whereas] present in test data point [True]
483 Text feature [deletion] present in test data point [True]
496 Text feature [suggested] present in test data point [True]
498 Text feature [genetic] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

4.3.3.2. For Incorrectly classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0341 0.0208 0.0116 0.0983 0.0237 0.0072 0.8005 0.0016 0.0022]]
Actual Class : 7
--------------------------------------------------
28 Text feature [constitutive] present in test data point [True]
32 Text feature [downstream] present in test data point [True]
35 Text feature [activation] present in test data point [True]
38 Text feature [enhanced] present in test data point [True]
40 Text feature [activated] present in test data point [True]
48 Text feature [2a] present in test data point [True]
52 Text feature [presence] present in test data point [True]
58 Text feature [activate] present in test data point [True]
59 Text feature [3b] present in test data point [True]
63 Text feature [erk] present in test data point [True]
64 Text feature [constitutively] present in test data point [True]
65 Text feature [transforming] present in test data point [True]
68 Text feature [activating] present in test data point [True]
69 Text feature [increased] present in test data point [True]
72 Text feature [concentrations] present in test data point [True]
75 Text feature [expressing] present in test data point [True]
76 Text feature [transformed] present in test data point [True]
137 Text feature [total] present in test data point [True]
139 Text feature [factor] present in test data point [True]
140 Text feature [signaling] present in test data point [True]
169 Text feature [membrane] present in test data point [True]
173 Text feature [2b] present in test data point [True]
174 Text feature [additional] present in test data point [True]
176 Text feature [24] present in test data point [True]
190 Text feature [occur] present in test data point [True]
192 Text feature [sensitive] present in test data point [True]
196 Text feature [years] present in test data point [True]
198 Text feature [mutant] present in test data point [True]
208 Text feature [download] present in test data point [True]
211 Text feature [long] present in test data point [True]
214 Text feature [transduced] present in test data point [True]
215 Text feature [signals] present in test data point [True]
226 Text feature [14] present in test data point [True]
232 Text feature [gfp] present in test data point [True]
237 Text feature [reverse] present in test data point [True]
238 Text feature [medium] present in test data point [True]
239 Text feature [days] present in test data point [True]
253 Text feature [pathways] present in test data point [True]
256 Text feature [positive] present in test data point [True]
259 Text feature [gain] present in test data point [True]
261 Text feature [common] present in test data point [True]
264 Text feature [provided] present in test data point [True]
268 Text feature [hr] present in test data point [True]
282 Text feature [cells] present in test data point [True]
285 Text feature [mutants] present in test data point [True]
289 Text feature [25] present in test data point [True]
294 Text feature [phosphorylation] present in test data point [True]
312 Text feature [previously] present in test data point [True]
314 Text feature [sensitivity] present in test data point [True]
316 Text feature [absence] present in test data point [True]
318 Text feature [domain] present in test data point [True]
319 Text feature [factors] present in test data point [True]
323 Text feature [transformation] present in test data point [True]
324 Text feature [inhibitor] present in test data point [True]
327 Text feature [inhibition] present in test data point [True]
330 Text feature [strongly] present in test data point [True]
343 Text feature [blot] present in test data point [True]
350 Text feature [independent] present in test data point [True]
351 Text feature [leukemia] present in test data point [True]
353 Text feature [showed] present in test data point [True]
359 Text feature [regulated] present in test data point [True]
```

```
360 Text feature [found] present in test data point [True]
361 Text feature [genomic] present in test data point [True]
362 Text feature [phospho] present in test data point [True]
373 Text feature [express] present in test data point [True]
379 Text feature [without] present in test data point [True]
381 Text feature [still] present in test data point [True]
382 Text feature [survival] present in test data point [True]
383 Text feature [overexpression] present in test data point [True]
387 Text feature [tyrosine] present in test data point [True]
405 Text feature [constructs] present in test data point [True]
406 Text feature [kit] present in test data point [True]
409 Text feature [four] present in test data point [True]
410 Text feature [properties] present in test data point [True]
412 Text feature [together] present in test data point [True]
414 Text feature [tumor] present in test data point [True]
419 Text feature [suggest] present in test data point [True]
422 Text feature [recently] present in test data point [True]
430 Text feature [assessed] present in test data point [True]
432 Text feature [whole] present in test data point [True]
433 Text feature [oncogenic] present in test data point [True]
435 Text feature [50] present in test data point [True]
441 Text feature [ba] present in test data point [True]
451 Text feature [indicate] present in test data point [True]
462 Text feature [frequency] present in test data point [True]
463 Text feature [f3] present in test data point [True]
470 Text feature [kinase] present in test data point [True]
471 Text feature [49] present in test data point [True]
474 Text feature [events] present in test data point [True]
475 Text feature [vector] present in test data point [True]
477 Text feature [within] present in test data point [True]
482 Text feature [20] present in test data point [True]
483 Text feature [bp] present in test data point [True]
485 Text feature [current] present in test data point [True]
489 Text feature [whereas] present in test data point [True]
Out of the top  500  features  95 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [0]:

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

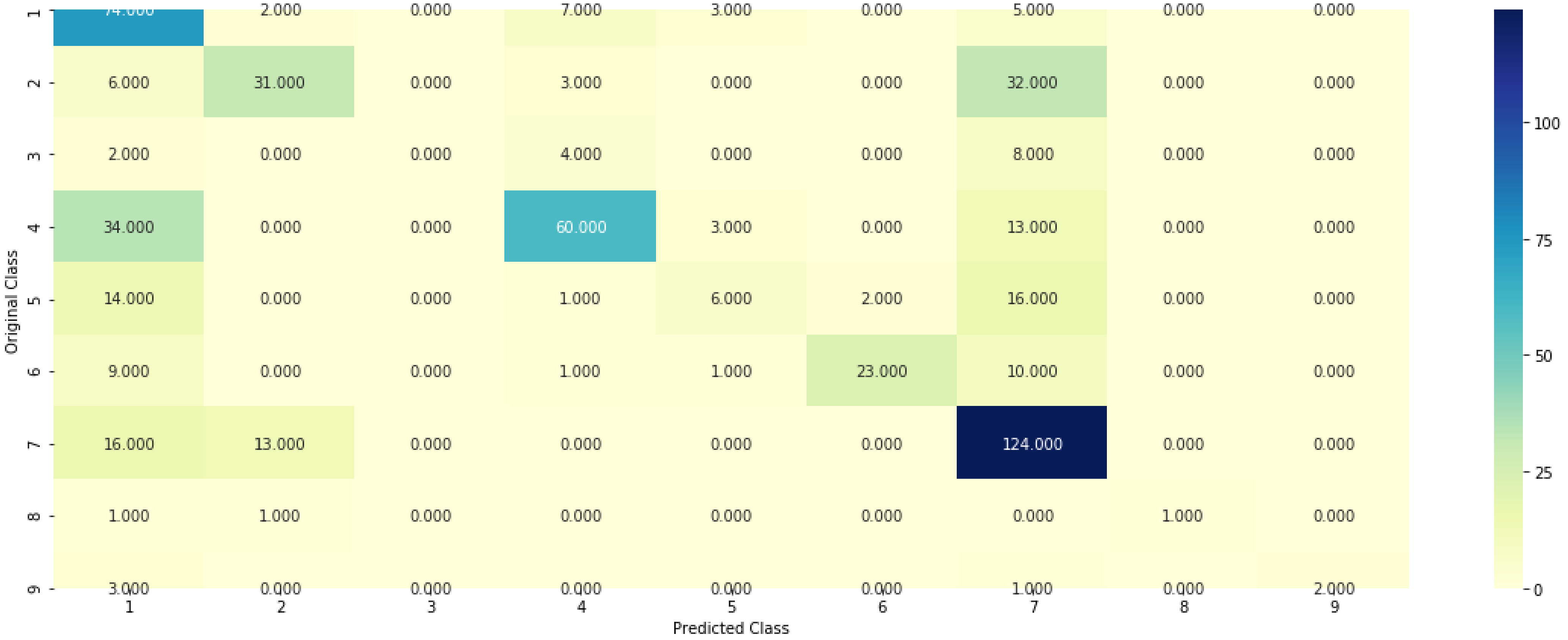
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1904249615739206
for n_estimators = 100 and max depth =  10
Log Loss : 1.2451128321174199
for n_estimators = 200 and max depth =  5
Log Loss : 1.1743557548536752
for n_estimators = 200 and max depth =  10
Log Loss : 1.2433256077452446
for n_estimators = 500 and max depth =  5
Log Loss : 1.1801290571203342
for n_estimators = 500 and max depth =  10
Log Loss : 1.2403635090127911
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1778212438881979
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2354165335958598
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1770848452168263
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2374077409416917
For values of best estimator =  200 The train log loss is: 0.8741010742392036
For values of best estimator =  200 The cross validation log loss is: 1.1743557548536754
For values of best estimator =  200 The test log loss is: 1.2077810143158516
```

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

In [0]:

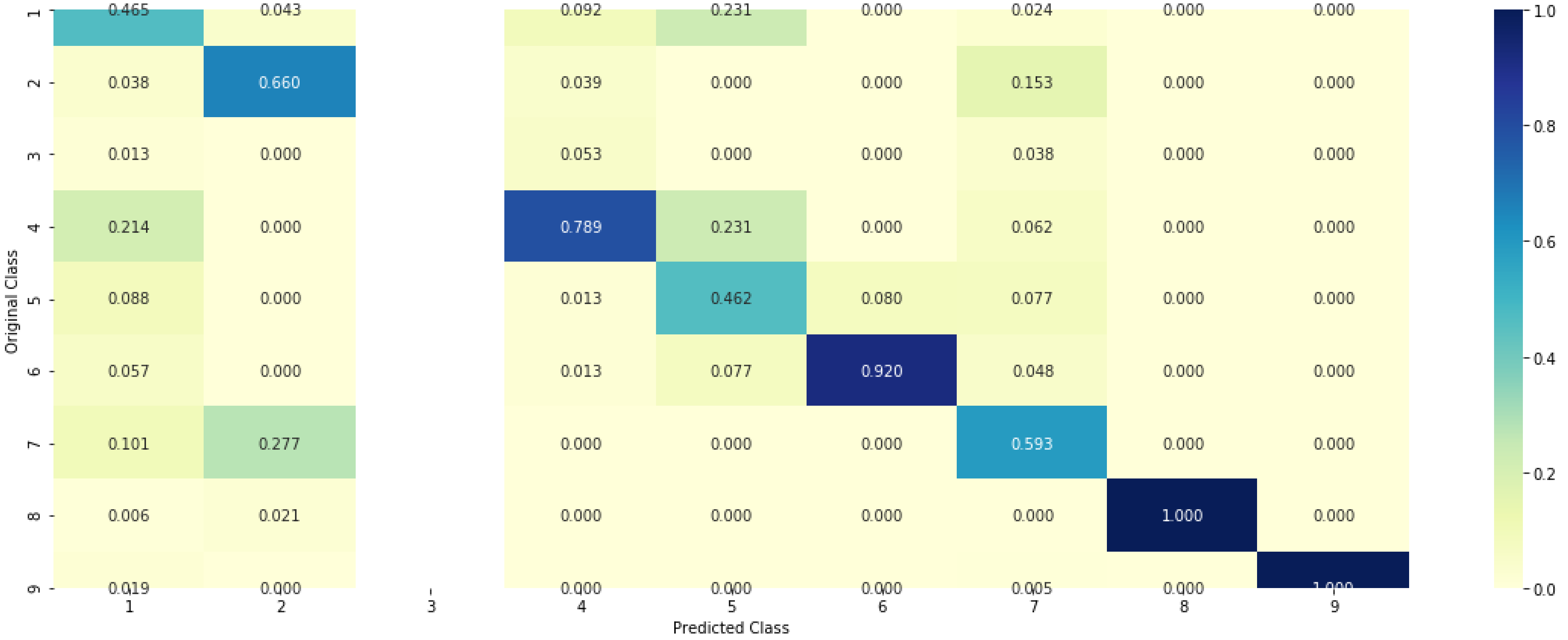
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, 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 : 1.17435575485 36752
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```



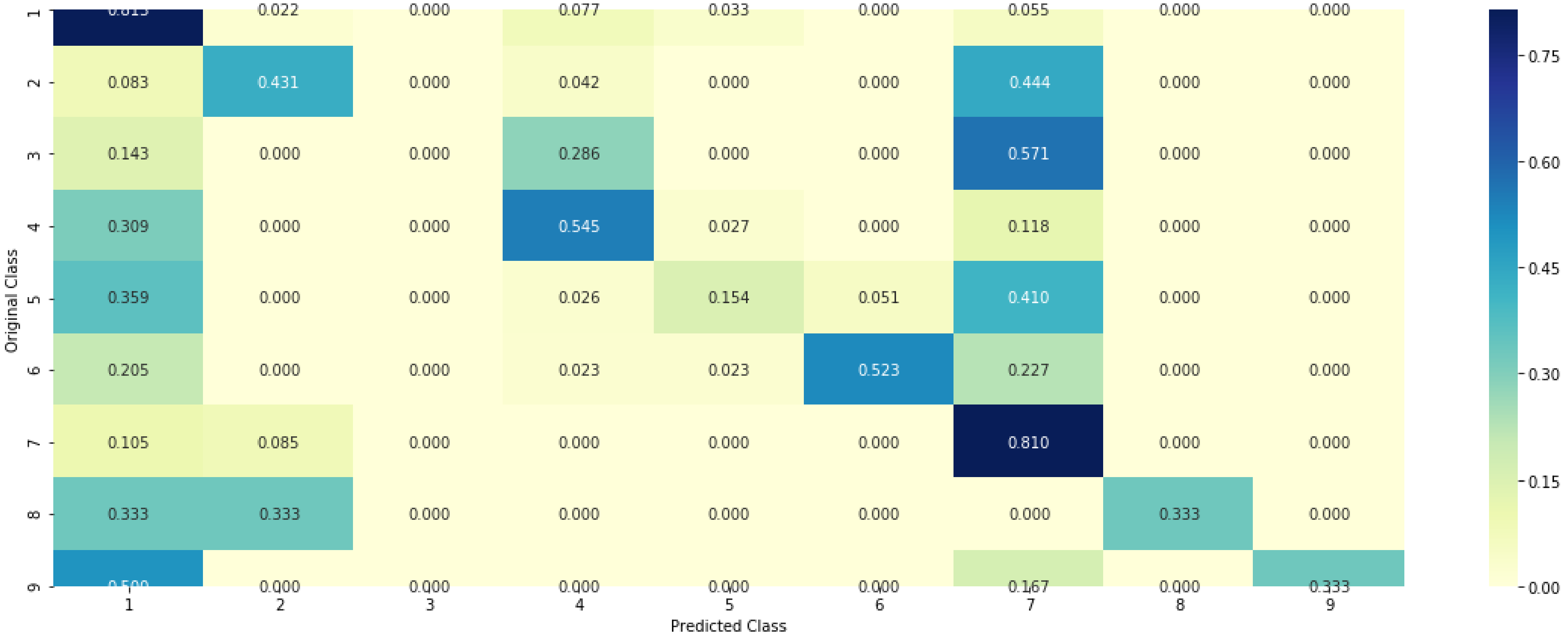
	1	2	3	4	5	6	7	8	9
1	74.000	2.000	0.000	7.000	3.000	0.000	5.000	0.000	0.000
2	6.000	31.000	0.000	3.000	0.000	0.000	32.000	0.000	0.000
3	2.000	0.000	0.000	4.000	0.000	0.000	8.000	0.000	0.000
4	34.000	0.000	0.000	60.000	3.000	0.000	13.000	0.000	0.000
5	14.000	0.000	0.000	1.000	6.000	2.000	16.000	0.000	0.000
6	9.000	0.000	0.000	1.000	1.000	23.000	10.000	0.000	0.000
7	16.000	13.000	0.000	0.000	0.000	0.000	124.000	0.000	0.000
8	1.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
9	3.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.465	0.043		0.092	0.231	0.000	0.024	0.000	0.000
2	0.038	0.660		0.039	0.000	0.000	0.153	0.000	0.000
3	0.013	0.000		0.053	0.000	0.000	0.038	0.000	0.000
4	0.214	0.000		0.789	0.231	0.000	0.062	0.000	0.000
5	0.088	0.000		0.013	0.462	0.080	0.077	0.000	0.000
6	0.057	0.000		0.013	0.077	0.920	0.048	0.000	0.000
7	0.101	0.277		0.000	0.000	0.000	0.593	0.000	0.000
8	0.006	0.021		0.000	0.000	0.000	0.000	1.000	0.000
9	0.019	0.000		0.000	0.000	0.000	0.005	0.000	1.000

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



In [21]:

```
# Almost 0 improvement
table.add_row(['All feature One-Hot','Random Forests', 1.17, 0.0])
print(table)
```

```
+---------------------+---------------+--------------+------------+
|      Vectorizer     |     Model     | Test Log Loss | Improvement |
+---------------------+---------------+--------------+------------+
|         NAN         |     Random    |     2.48     |     0      |
| Gene Feature One-Hot |   Linear SVM  |     1.19     |    0.01    |
| Var Feature One-Hot |   Linear SVM  |     1.73     |    0.0     |
| Text Feature One-Hot |   Linear SVM  |     1.1      |    0.09    |
|  TfIdf All Features |  Naive Bayes  |     1.17     |    0.1     |
| All feature One-Hot |      kNN      |     0.98     |    0.12    |
| All feature One-Hot |    Logistic   |     0.97     |    0.24    |
| All feature One-Hot |  Logistic Reg |     1.01     |    0.24    |
| All feature One-Hot |      kNN      |     1.06     |    0.18    |
| All feature One-Hot | Random Forests |     1.17     |    0.0     |
+---------------------+---------------+--------------+------------+
```

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
In [0]: # test_point_index = 10
        clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],
                          criterion='gini', max_depth=max_depth[int(best_alpha%2)],
                          random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.186  0.058  0.0175 0.545  0.0606 0.0608 0.058  0.006  0.0082]]
Actual Class : 4
--------------------------------------------------
2 Text feature [inhibitors] present in test data point [True]
7 Text feature [suppressor] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [kinases] present in test data point [True]
16 Text feature [protein] present in test data point [True]
28 Text feature [proteins] present in test data point [True]
32 Text feature [inhibited] present in test data point [True]
36 Text feature [growth] present in test data point [True]
39 Text feature [cells] present in test data point [True]
62 Text feature [cell] present in test data point [True]
66 Text feature [catalytic] present in test data point [True]
74 Text feature [dna] present in test data point [True]
83 Text feature [families] present in test data point [True]
86 Text feature [activity] present in test data point [True]
87 Text feature [proliferation] present in test data point [True]
92 Text feature [many] present in test data point [True]
95 Text feature [genes] present in test data point [True]
Out of the top  100  features  17 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [0]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0695 0.2286 0.0224 0.0676 0.0474 0.0467 0.5083 0.0054 0.0042]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [inhibitors] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
5 Text feature [constitutive] present in test data point [True]
6 Text feature [treatment] present in test data point [True]
8 Text feature [activated] present in test data point [True]
9 Text feature [function] present in test data point [True]
10 Text feature [loss] present in test data point [True]
11 Text feature [tyrosine] present in test data point [True]
13 Text feature [inhibitor] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [kinases] present in test data point [True]
16 Text feature [protein] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
22 Text feature [erk] present in test data point [True]
24 Text feature [constitutively] present in test data point [True]
25 Text feature [oncogenic] present in test data point [True]
26 Text feature [functional] present in test data point [True]
28 Text feature [proteins] present in test data point [True]
29 Text feature [transforming] present in test data point [True]
31 Text feature [expression] present in test data point [True]
36 Text feature [growth] present in test data point [True]
37 Text feature [patients] present in test data point [True]
39 Text feature [cells] present in test data point [True]
44 Text feature [treated] present in test data point [True]
45 Text feature [receptor] present in test data point [True]
54 Text feature [phosphatase] present in test data point [True]
55 Text feature [activate] present in test data point [True]
59 Text feature [pten] present in test data point [True]
60 Text feature [clinical] present in test data point [True]
61 Text feature [sensitivity] present in test data point [True]
62 Text feature [cell] present in test data point [True]
65 Text feature [sensitive] present in test data point [True]
67 Text feature [f3] present in test data point [True]
69 Text feature [downstream] present in test data point [True]
70 Text feature [expressing] present in test data point [True]
71 Text feature [likely] present in test data point [True]
74 Text feature [dna] present in test data point [True]
75 Text feature [predicted] present in test data point [True]
78 Text feature [ic50] present in test data point [True]
79 Text feature [inhibition] present in test data point [True]
86 Text feature [activity] present in test data point [True]
87 Text feature [proliferation] present in test data point [True]
88 Text feature [combined] present in test data point [True]
94 Text feature [presence] present in test data point [True]
95 Text feature [genes] present in test data point [True]
98 Text feature [events] present in test data point [True]
Out of the top  100  features  49 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [0]:

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

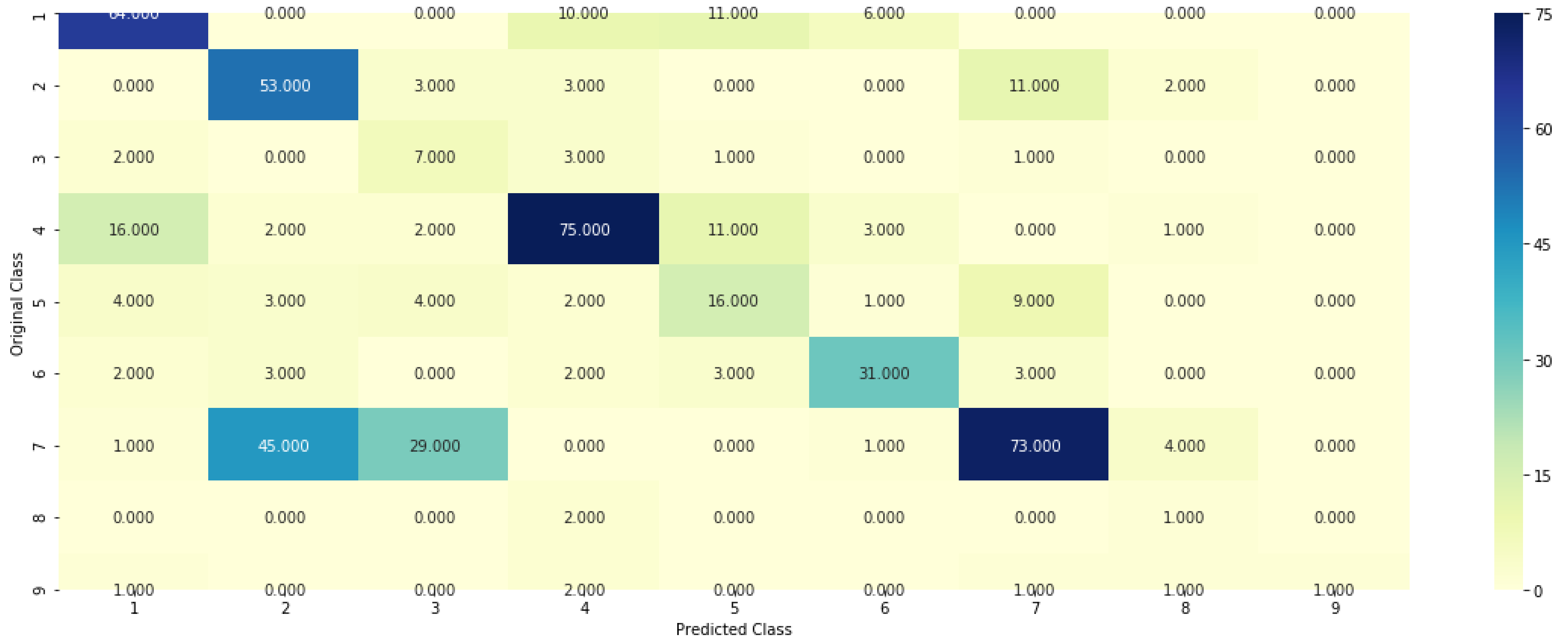
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.08799955346638
for n_estimators = 10 and max depth =  3
Log Loss : 1.6292598048854747
for n_estimators = 10 and max depth =  5
Log Loss : 1.5538462026849176
for n_estimators = 10 and max depth =  10
Log Loss : 1.7651267622285174
for n_estimators = 50 and max depth =  2
Log Loss : 1.7095120082740844
for n_estimators = 50 and max depth =  3
Log Loss : 1.3609680193984586
for n_estimators = 50 and max depth =  5
Log Loss : 1.3282335156087746
for n_estimators = 50 and max depth =  10
Log Loss : 1.686563481882049
for n_estimators = 100 and max depth =  2
Log Loss : 1.4991066763869016
for n_estimators = 100 and max depth =  3
Log Loss : 1.381282330726151
for n_estimators = 100 and max depth =  5
Log Loss : 1.2587033937246284
for n_estimators = 100 and max depth =  10
Log Loss : 1.7016512450326247
for n_estimators = 200 and max depth =  2
Log Loss : 1.558684961945638
for n_estimators = 200 and max depth =  3
Log Loss : 1.4089652899285743
for n_estimators = 200 and max depth =  5
Log Loss : 1.286880681196974
for n_estimators = 200 and max depth =  10
Log Loss : 1.6243484455214874
for n_estimators = 500 and max depth =  2
Log Loss : 1.618108149915873
for n_estimators = 500 and max depth =  3
Log Loss : 1.4786062299374105
for n_estimators = 500 and max depth =  5
Log Loss : 1.3204994379164592
for n_estimators = 500 and max depth =  10
Log Loss : 1.6986766308404153
for n_estimators = 1000 and max depth =  2
Log Loss : 1.603642813863514
for n_estimators = 1000 and max depth =  3
Log Loss : 1.497693696592617
for n_estimators = 1000 and max depth =  5
Log Loss : 1.325998879678907
for n_estimators = 1000 and max depth =  10
Log Loss : 1.676514615565835
For values of best alpha =  100 The train log loss is: 0.05163449977779097
For values of best alpha =  100 The cross validation log loss is: 1.2587033937246284
For values of best alpha =  100 The test log loss is: 1.3752953394519125
```

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

In [0]:

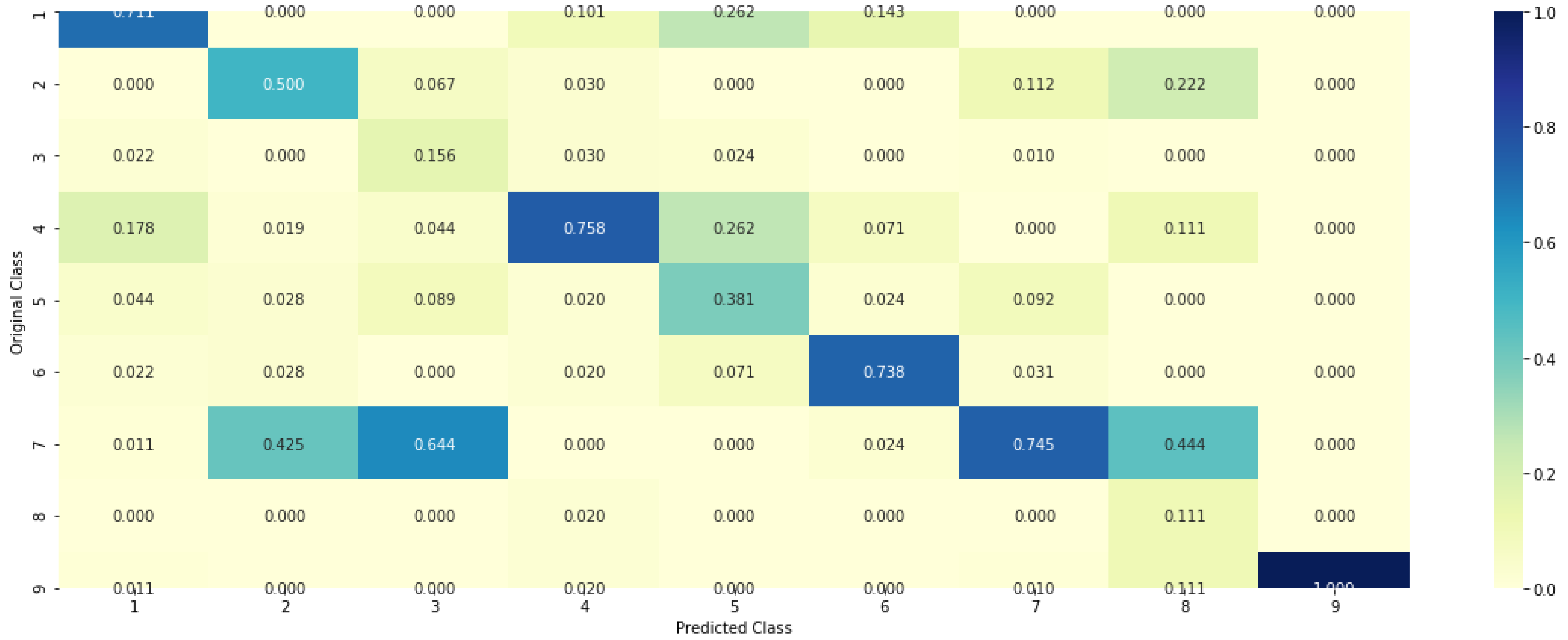
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, 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 : 1.2587033937246284
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```



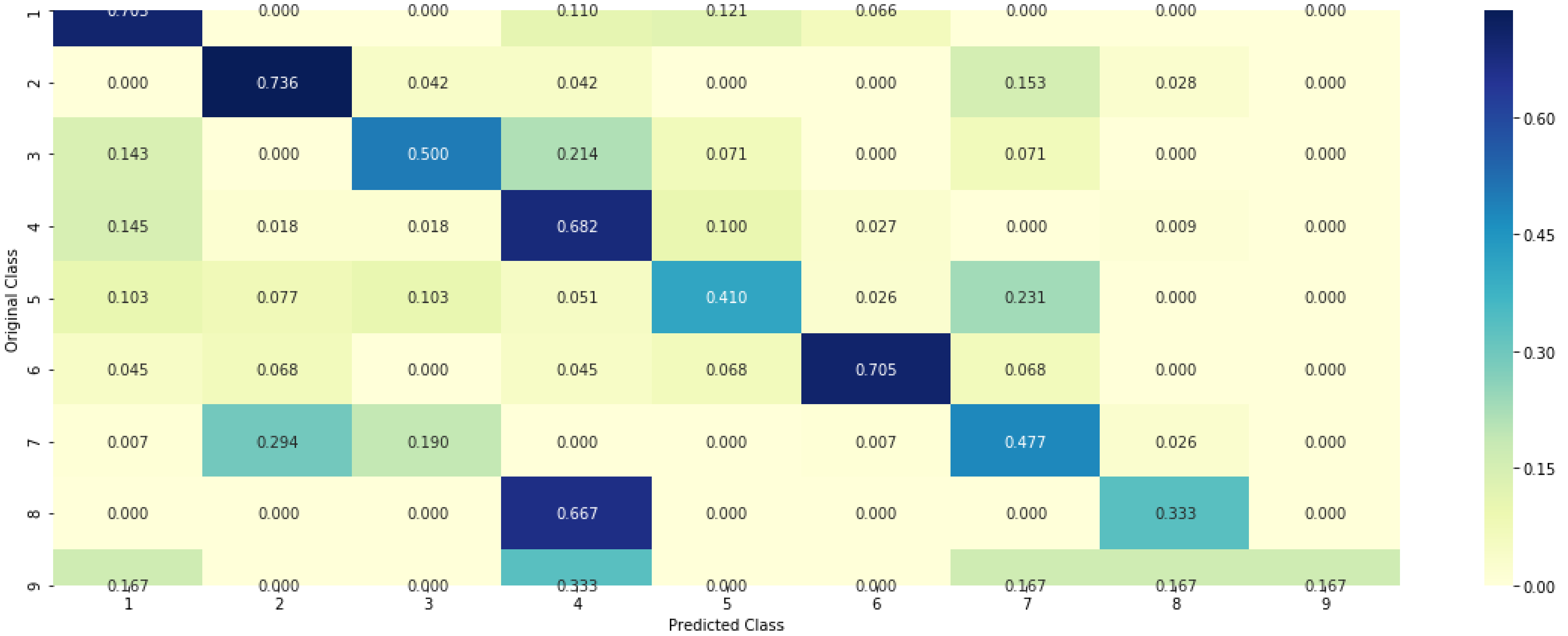
	1	2	3	4	5	6	7	8	9
1	64.000	0.000	0.000	10.000	11.000	6.000	0.000	0.000	0.000
2	0.000	53.000	3.000	3.000	0.000	0.000	11.000	2.000	0.000
3	2.000	0.000	7.000	3.000	1.000	0.000	1.000	0.000	0.000
4	16.000	2.000	2.000	75.000	11.000	3.000	0.000	1.000	0.000
5	4.000	3.000	4.000	2.000	16.000	1.000	9.000	0.000	0.000
6	2.000	3.000	0.000	2.000	3.000	31.000	3.000	0.000	0.000
7	1.000	45.000	29.000	0.000	0.000	1.000	73.000	4.000	0.000
8	0.000	0.000	0.000	2.000	0.000	0.000	0.000	1.000	0.000
9	1.000	0.000	0.000	2.000	0.000	0.000	1.000	1.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.711	0.000	0.000	0.101	0.262	0.143	0.000	0.000	0.000
2	0.000	0.500	0.067	0.030	0.000	0.000	0.112	0.222	0.000
3	0.022	0.000	0.156	0.030	0.024	0.000	0.010	0.000	0.000
4	0.178	0.019	0.044	0.758	0.262	0.071	0.000	0.111	0.000
5	0.044	0.028	0.089	0.020	0.381	0.024	0.092	0.000	0.000
6	0.022	0.028	0.000	0.020	0.071	0.738	0.031	0.000	0.000
7	0.011	0.425	0.644	0.000	0.000	0.024	0.745	0.444	0.000
8	0.000	0.000	0.000	0.020	0.000	0.000	0.000	0.111	0.000
9	0.011	0.000	0.000	0.020	0.000	0.000	0.010	0.111	1.000

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



In [23]:

```
# This improved a lot - 0.16
table.add_row(['All feat Res Encod','Random Forests', 1.25, 0.16])
print(table)
```

```
+---------------------+---------------+----------------+-------------+
|      Vectorizer     |     Model     | Test Log Loss  | Improvement |
+---------------------+---------------+----------------+-------------+
|         NAN         |     Random    |      2.48      |      0      |
| Gene Feature One-Hot |   Linear SVM  |      1.19      |     0.01    |
| Var Feature One-Hot |   Linear SVM  |      1.73      |     0.0     |
| Text Feature One-Hot |   Linear SVM  |      1.1       |     0.09    |
|  TfIdf All Features |  Naive Bayes  |      1.17      |     0.1     |
| All feature One-Hot |      kNN      |      0.98      |     0.12    |
| All feature One-Hot |    Logistic   |      0.97      |     0.24    |
| All feature One-Hot |  Logistic Reg |      1.01      |     0.24    |
| All feature One-Hot |      kNN      |      1.06      |     0.18    |
| All feature One-Hot | Random Forests |      1.17      |     0.0     |
|  All feat Res Encod | Random Forests |      1.25      |     0.16    |
+---------------------+---------------+----------------+-------------+
```

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: [[0.054  0.0409 0.161  0.6027 0.0254 0.0435 0.0143 0.037  0.0212]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
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
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.019  0.1148 0.2548 0.0213 0.0292 0.0526 0.4499 0.0428 0.0154]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
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
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [0]:

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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


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

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

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

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


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

```
sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
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 Classifer : 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
```

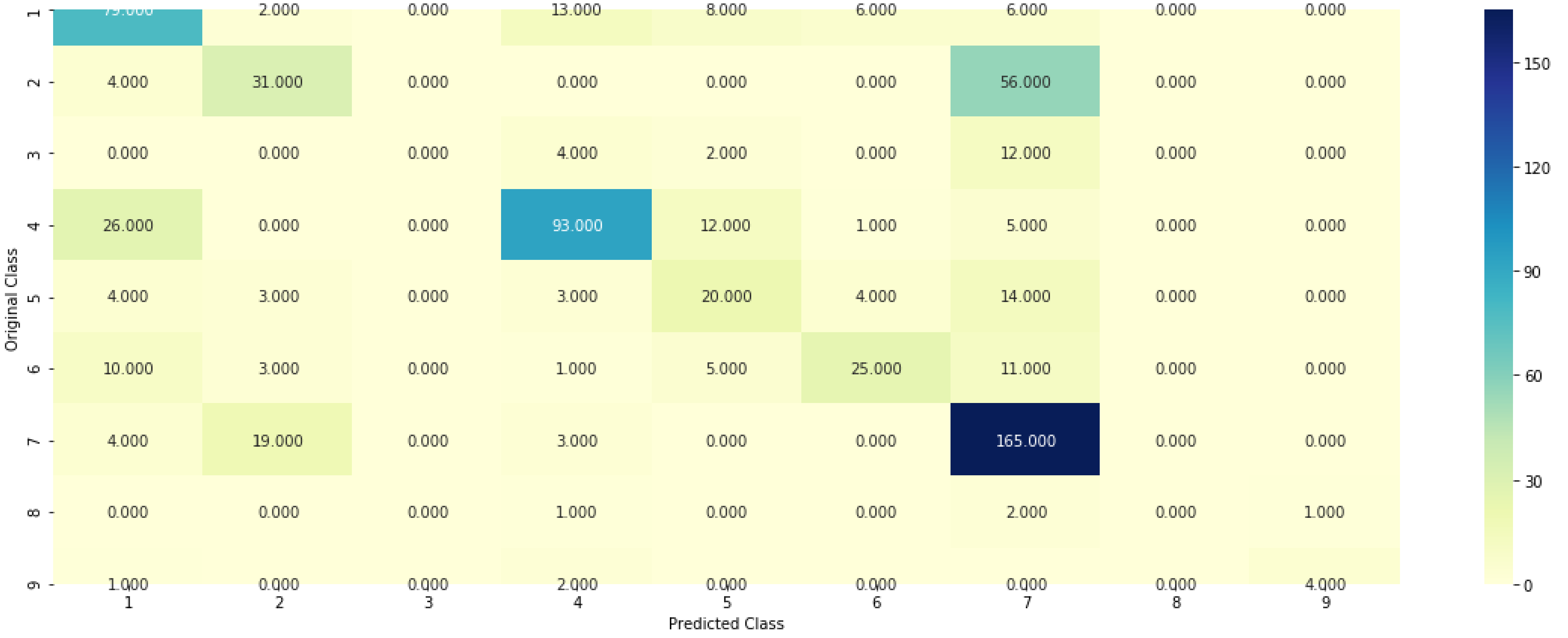
```
Logistic Regression :  Log Loss: 0.99
Support vector machines : Log Loss: 1.77
Naive Bayes : Log Loss: 1.11
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.034
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.491
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.068
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.038
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.130
```

4.7.2 testing the model with the best hyper parameters

In [0]:

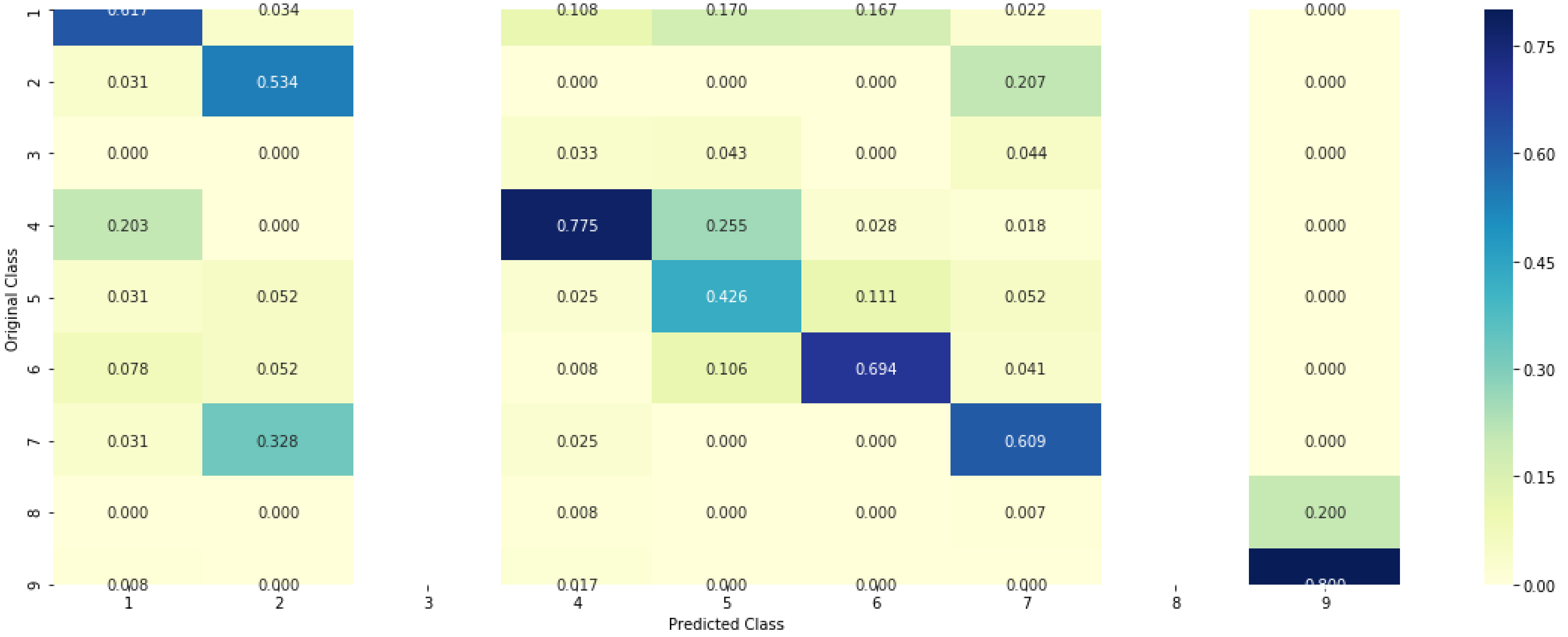
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.8216929455191615
Log loss (CV) on the stacking classifier : 1.067617466149384
Log loss (test) on the stacking classifier : 1.1459527182678662
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```



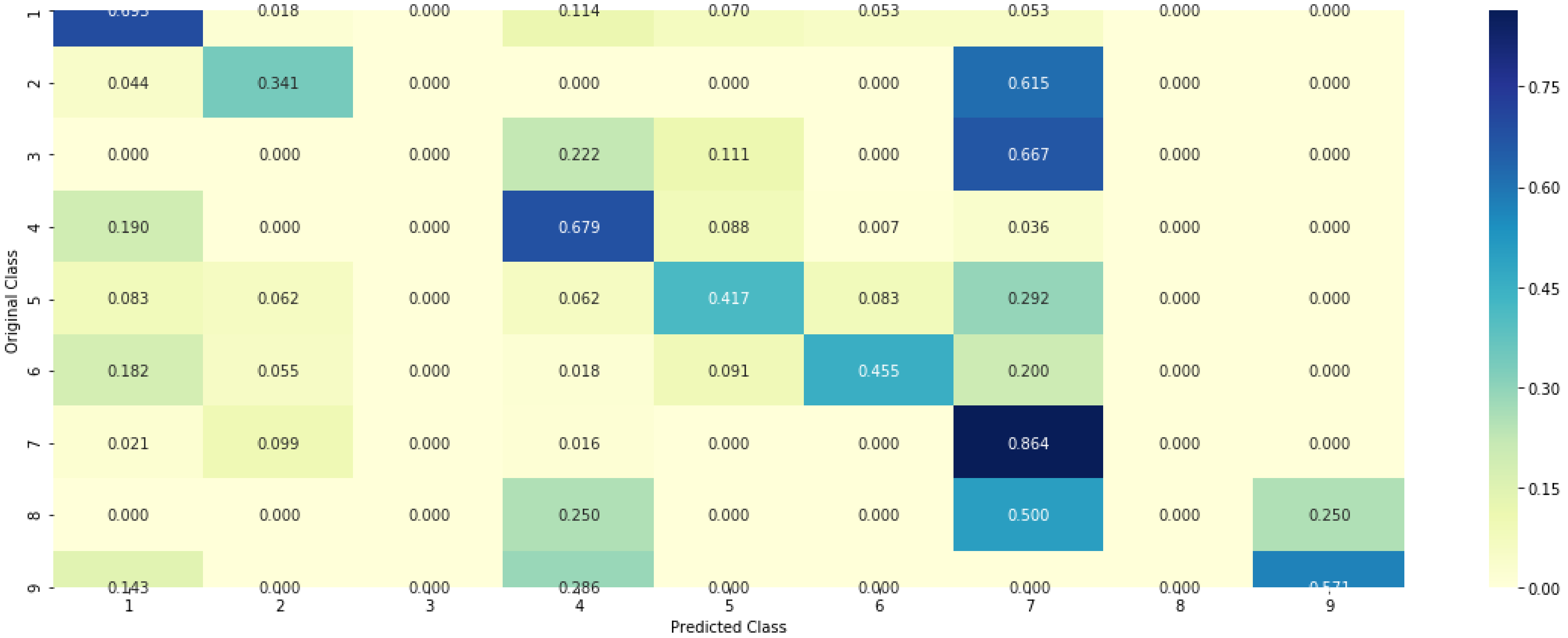
Original Class	1	2	3	4	5	6	7	8	9
1	79.000	2.000	0.000	13.000	8.000	6.000	6.000	0.000	0.000
2	4.000	31.000	0.000	0.000	0.000	0.000	56.000	0.000	0.000
3	0.000	0.000	0.000	4.000	2.000	0.000	12.000	0.000	0.000
4	26.000	0.000	0.000	93.000	12.000	1.000	5.000	0.000	0.000
5	4.000	3.000	0.000	3.000	20.000	4.000	14.000	0.000	0.000
6	10.000	3.000	0.000	1.000	5.000	25.000	11.000	0.000	0.000
7	4.000	19.000	0.000	3.000	0.000	0.000	165.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	1.000
9	1.000	0.000	0.000	2.000	0.000	0.000	0.000	0.000	4.000

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



Original Class	1	2	3	4	5	6	7	8	9
1	0.617	0.034		0.108	0.170	0.167	0.022		0.000
2	0.031	0.534		0.000	0.000	0.000	0.207		0.000
3	0.000	0.000		0.033	0.043	0.000	0.044		0.000
4	0.203	0.000		0.775	0.255	0.028	0.018		0.000
5	0.031	0.052		0.025	0.426	0.111	0.052		0.000
6	0.078	0.052		0.008	0.106	0.694	0.041		0.000
7	0.031	0.328		0.025	0.000	0.000	0.609		0.000
8	0.000	0.000		0.008	0.000	0.000	0.007		0.200
9	0.008	0.000		0.017	0.000	0.000	0.000		0.800

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



In [25]:

```
# There was very little improvement in the stacking classifiers
table.add_row(['All feat One-hot','Stacking Classifier', 1.14, 0.01])
print(table)
```

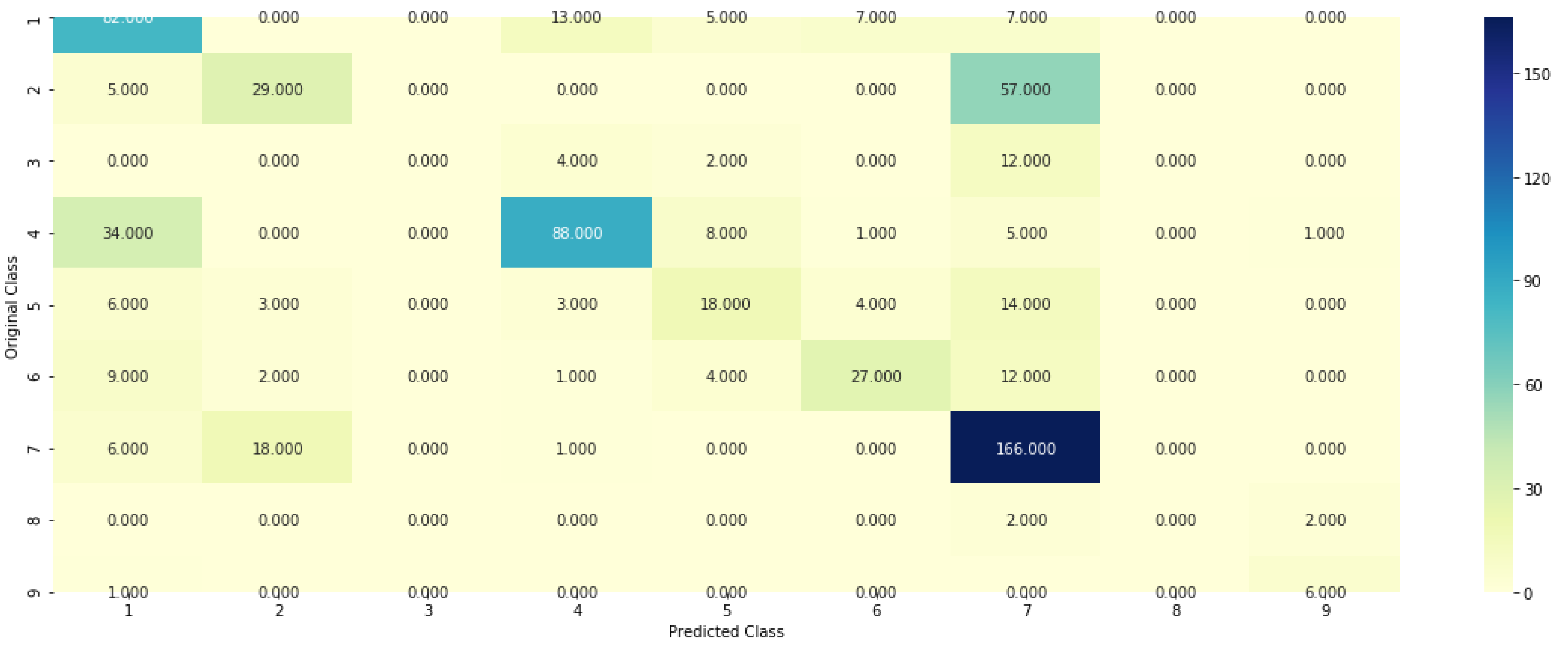
```
+---------------------+--------------------+---------------+-------------+
|      Vectorizer     |       Model        | Test Log Loss | Improvement |
+---------------------+--------------------+---------------+-------------+
|         NAN         |       Random       |      2.48     |      0      |
| Gene Feature One-Hot |     Linear SVM     |      1.19     |     0.01    |
| Var Feature One-Hot |     Linear SVM     |      1.73     |     0.0     |
| Text Feature One-Hot |     Linear SVM     |      1.1      |     0.09    |
|  TfIdf All Features |    Naive Bayes     |      1.17     |     0.1     |
| All feature One-Hot |        kNN         |      0.98     |     0.12    |
| All feature One-Hot |      Logistic      |      0.97     |     0.24    |
| All feature One-Hot |    Logistic Reg    |      1.01     |     0.24    |
| All feature One-Hot |        kNN         |      1.06     |     0.18    |
| All feature One-Hot |   Random Forests   |      1.17     |     0.0     |
|  All feat Res Encod |   Random Forests   |      1.25     |     0.16    |
|   All feat One-hot  | Stacking Classifier |      1.14     |     0.01    |
+---------------------+--------------------+---------------+-------------+
```

4.7.3 Maximum Voting classifier

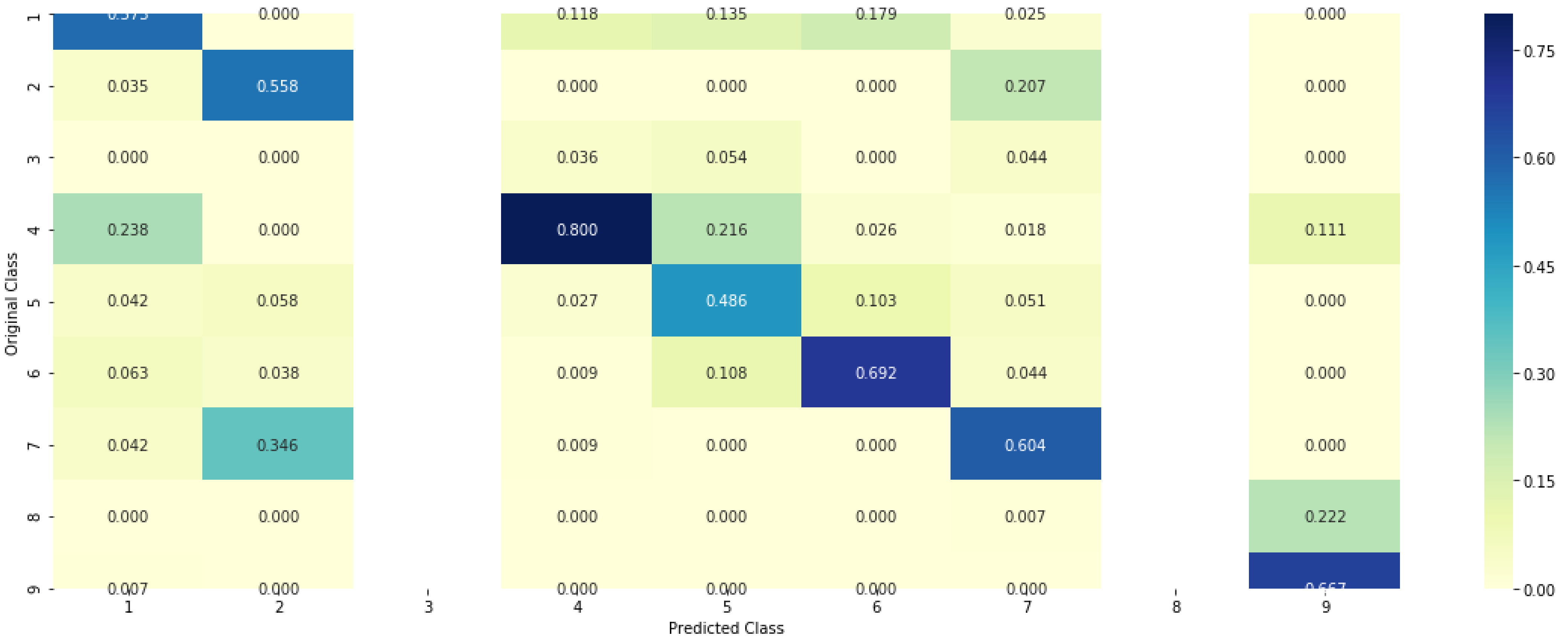
In [0]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2),
                                    ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
                                vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y,
                                vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
                                vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :",
    np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

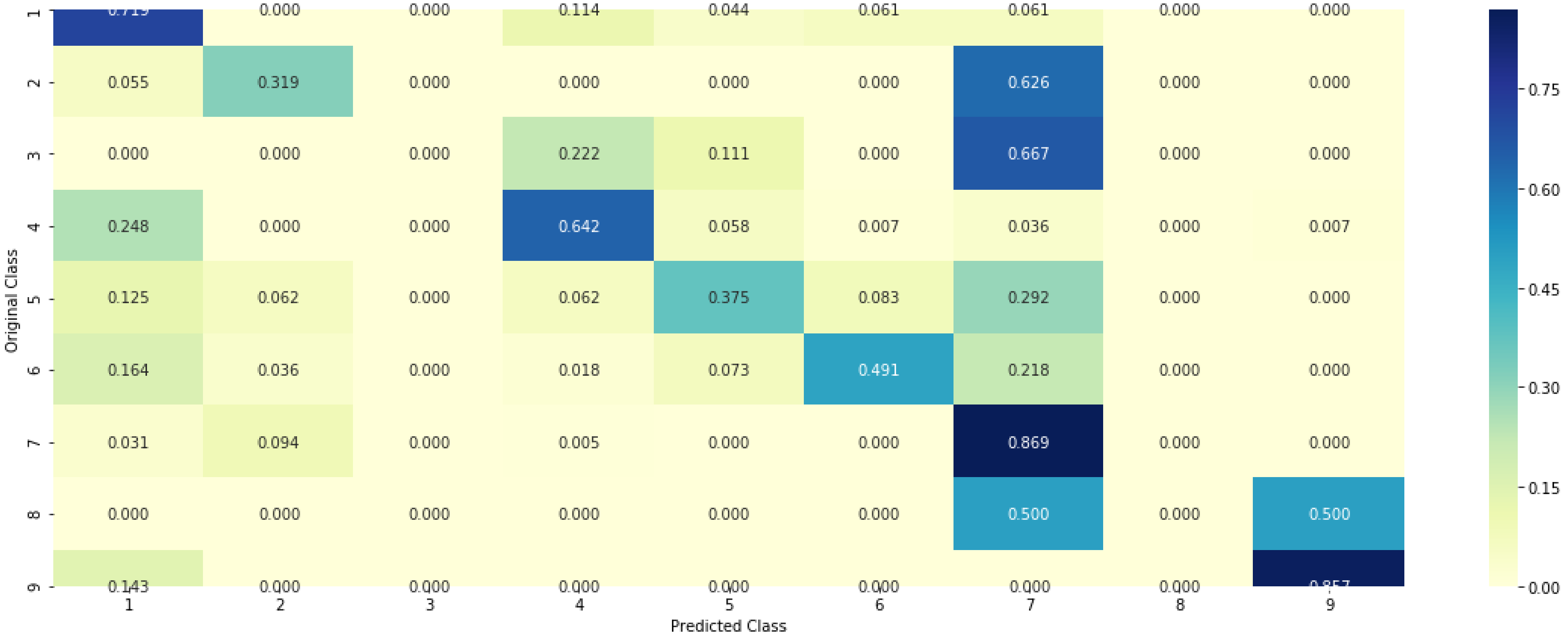
```
Log loss (train) on the VotingClassifier : 0.9576933926044386
Log loss (CV) on the VotingClassifier : 1.1446572769408272
Log loss (test) on the VotingClassifier : 1.1985546398439435
Number of missclassified point : 0.3744360902255639
-------------------- Confusion matrix --------------------
```



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



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



In [27]:

```
# We see a slight improvement - 0.03
table.add_row(['All feat One Hot','Max Voting', 1.19, 0.03])
print(table)
```

```
+----------------------+--------------------+---------------+-------------+
|      Vectorizer      |       Model        | Test Log Loss | Improvement |
+----------------------+--------------------+---------------+-------------+
|         NAN          |       Random       |      2.48     |      0      |
| Gene Feature One-Hot |     Linear SVM     |      1.19     |     0.01    |
| Var Feature One-Hot  |     Linear SVM     |      1.73     |     0.0     |
| Text Feature One-Hot |     Linear SVM     |      1.1      |     0.09    |
|  TfIdf All Features  |    Naive Bayes     |      1.17     |     0.1     |
| All feature One-Hot  |        kNN         |      0.98     |     0.12    |
| All feature One-Hot  |      Logistic      |      0.97     |     0.24    |
| All feature One-Hot  |    Logistic Reg    |      1.01     |     0.24    |
| All feature One-Hot  |        kNN         |      1.06     |     0.18    |
| All feature One-Hot  |   Random Forests   |      1.17     |     0.0     |
|  All feat Res Encod  |   Random Forests   |      1.25     |     0.16    |
|   All feat One-hot   | Stacking Classifier|      1.14     |     0.01    |
|   All feat One Hot   |     Max Voting     |      1.19     |     0.03    |
+----------------------+--------------------+---------------+-------------+
```

5. Assignments

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

In [0]:

```python
# Reading Data
data = pd.read_csv(path+'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

In [0]:

```python
# note the seprator in this file
data_text =pd.read_csv(path+"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...

In [0]:

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

            for word in sent.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
                          # data.loc[index,'Gene'].replace('\s+', '_')+\
                          # " "+data.loc[index,'Variation'].replace('\s+', '_')
```

In [0]:

```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing_custom(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 : 11.973772999999937 seconds
```

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 orphan cyclin product fam58a whose muta...
1	1	CBL	W802*	2	c cbl e3 ubiquitin ligase adaptor molecule imp...
2	2	CBL	Q249E	2	c cbl e3 ubiquitin ligase adaptor molecule imp...
3	3	CBL	N454D	3	cbl mutations identified 3 cases 11q aupd anal...
4	4	CBL	L399V	4	several human c cbl cbl structures recently so...

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	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

In [0]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [0]:
```
y_true = result['Class'].values
# result.drop(columns=['ID','Gene','Variation'])
# split the data into test and train by maintaining same distribution of
# output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true,
                                        stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining
# same distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train,
                                        stratify=y_train, test_size=0.2)
```

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
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]:
```
print(train_gene_feature_onehotCoding.shape)
print(test_gene_feature_onehotCoding.shape)
print(cv_gene_feature_onehotCoding.shape)
```

```
(2124, 234)
(665, 234)
(532, 234)
```

In [0]:
```
print(gene_vectorizer.get_feature_names()[:10])
```

```
['abl1', 'acvr1', 'ago2', 'akt1', 'akt2', 'akt3', 'alk', 'ar', 'araf', 'arid1a']
```

In [0]:
```
# one-hot encoding of Gene feature.

gene_vectorizer_tfIdf = TfidfVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding_tfIdf = gene_vectorizer_tfIdf.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_tfIdf = gene_vectorizer_tfIdf.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_tfIdf = gene_vectorizer_tfIdf.transform(cv_df['Gene'])
```

In [0]:
```
print(train_gene_feature_onehotCoding_tfIdf.shape)
print(test_gene_feature_onehotCoding_tfIdf.shape)
print(cv_gene_feature_onehotCoding_tfIdf.shape)
```

```
(2124, 234)
(665, 234)
(532, 234)
```

In [0]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
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_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
(2124, 2122)
(665, 2122)
(532, 2122)
```

In [0]:

```
# one-hot encoding of variation feature.
variation_vectorizer_tfIdf = TfidfVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding_tfIdf = variation_vectorizer_tfIdf.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_tfIdf = variation_vectorizer_tfIdf.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_tfIdf = variation_vectorizer_tfIdf.transform(cv_df['Variation'])
```

In [0]:

```
print(train_variation_feature_onehotCoding_tfIdf.shape)
print(test_variation_feature_onehotCoding_tfIdf.shape)
print(cv_variation_feature_onehotCoding_tfIdf.shape)
```

```
(2124, 2122)
(665, 2122)
(532, 2122)
```

In [0]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,4),max_features=2000)
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_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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

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

```
Total number of unique words in train data : 2000
```

In [0]:

```
# building a TfIdfVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer_tfIdf = TfidfVectorizer(min_df=3,ngram_range=(1,4),max_features=2000)
train_text_feature_onehotCoding_tfIdf = text_vectorizer_tfIdf.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features_tfIdf= text_vectorizer_tfIdf.get_feature_names()
# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
train_text_fea_counts_tfIdf = train_text_feature_onehotCoding_tfIdf.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_tfIdf = dict(zip(list(train_text_features_tfIdf),train_text_fea_counts_tfIdf))

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

```
Total number of unique words in train data : 2000
```

In [0]:

```
# don't forget to normalize every feature
train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCoding, axis=0)
test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCoding, axis=0)
cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding, axis=0)
train_gene_feature_onehotCoding_tfIdf = normalize(train_gene_feature_onehotCoding_tfIdf, axis=0)
test_gene_feature_onehotCoding_tfIdf = normalize(test_gene_feature_onehotCoding_tfIdf, axis=0)
cv_gene_feature_onehotCoding_tfIdf = normalize(cv_gene_feature_onehotCoding_tfIdf, axis=0)


train_variation_feature_onehotCoding = normalize(train_variation_feature_onehotCoding, axis=0)
test_variation_feature_onehotCoding = normalize(test_variation_feature_onehotCoding, axis=0)
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_onehotCoding, axis=0)
train_variation_feature_onehotCoding_tfIdf = normalize(train_variation_feature_onehotCoding_tfIdf, axis=0)
test_variation_feature_onehotCoding_tfIdf = normalize(test_variation_feature_onehotCoding_tfIdf, axis=0)
cv_variation_feature_onehotCoding_tfIdf = normalize(cv_variation_feature_onehotCoding_tfIdf, axis=0)


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

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

In [0]:

```
print(train_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)
print(train_text_feature_onehotCoding_tfIdf.shape)
print(test_text_feature_onehotCoding_tfIdf.shape)
print(cv_text_feature_onehotCoding_tfIdf.shape)
```

```
(2124, 2000)
(665, 2000)
(532, 2000)
(2124, 2000)
(665, 2000)
(532, 2000)
```

In [0]:

```
# Stacking all the features
train_gene_var_onehotCoding = hstack((#train_gene_feature_onehotCoding,
                                      train_gene_feature_onehotCoding_tfIdf,
                                      #train_variation_feature_onehotCoding,
                                      train_variation_feature_onehotCoding_tfIdf
                                      ))
test_gene_var_onehotCoding = hstack((#test_gene_feature_onehotCoding,
                                     test_gene_feature_onehotCoding_tfIdf,
                                     #test_variation_feature_onehotCoding,
                                     test_variation_feature_onehotCoding_tfIdf
                                     ))
cv_gene_var_onehotCoding = hstack((#cv_gene_feature_onehotCoding,
                                   cv_gene_feature_onehotCoding_tfIdf,
                                   #cv_variation_feature_onehotCoding,
                                   cv_variation_feature_onehotCoding_tfIdf
                                   ))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,
                               # train_text_feature_onehotCoding,
                               train_text_feature_onehotCoding_tfIdf
                               )).tocsr()
train_y = np.array(list(train_df['Class']))

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

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

# train_x_onehotCoding = train_text_feature_onehotCoding_tfIdf.tocsr()
# train_y = np.array(list(train_df['Class']))
# test_x_onehotCoding = test_text_feature_onehotCoding_tfIdf.tocsr()
# test_y = np.array(list(test_df['Class']))
# cv_x_onehotCoding = cv_text_feature_onehotCoding_tfIdf.tocsr()
# cv_y = np.array(list(cv_df['Class']))
```

In [0]:

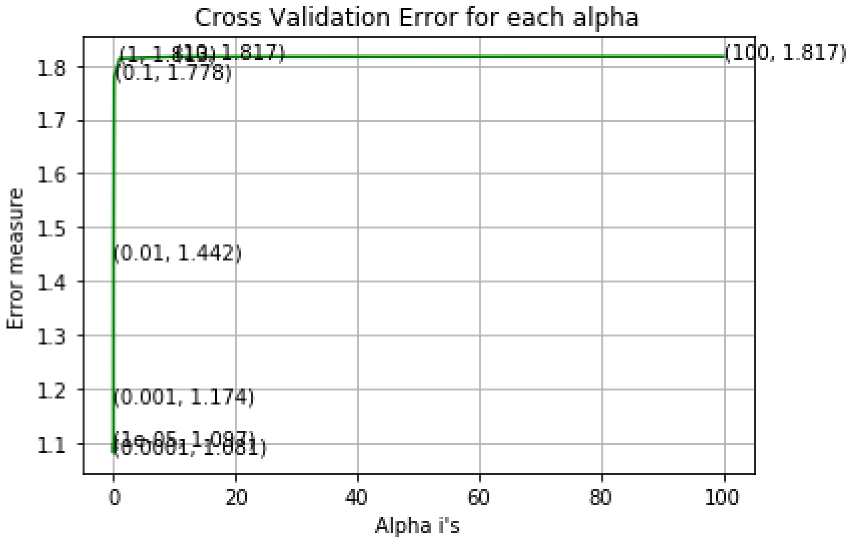
```
print(train_x_onehotCoding.shape)
print(test_x_onehotCoding.shape)
print(cv_x_onehotCoding.shape)
```

```
(2124, 4356)
(665, 4356)
(532, 4356)
```

In [0]:

```
# Train a Logistic regression+Calibration model using text features which are
# on-hot encoded BUT USING CountVectoriser and unigrams and bigrams
alpha = [10 ** x for x in range(-5, 3)]

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log',
                       random_state=42,class_weight='balanced')
    #clf = LogisticRegression(C=i, penalty='l2',random_state=42,
    #                         class_weight='balanced')
    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(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, predict_y,
                                       labels=clf.classes_,
                                       eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",
          log_loss(cv_y, 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_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(train_y, 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(cv_y, 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(test_y, predict_y,
                              labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.0966846557586194
For values of alpha =  0.0001 The log loss is: 1.0807408377942256
For values of alpha =  0.001 The log loss is: 1.1737465559195335
For values of alpha =  0.01 The log loss is: 1.4415711418361898
For values of alpha =  0.1 The log loss is: 1.7779385784922597
For values of alpha =  1 The log loss is: 1.8132064109706982
For values of alpha =  10 The log loss is: 1.8165693800903042
For values of alpha =  100 The log loss is: 1.816968822013241
```



```
For values of best alpha =  0.0001 The train log loss is: 0.428967305802904
For values of best alpha =  0.0001 The cross validation log loss is: 1.114801801762012
For values of best alpha =  0.0001 The test log loss is: 0.9851691737693713
```

In [29]:
```
# This improved a lot
table.add_row(['All feat One Hot','Logistic Reg', 0.98, 0.12])
print(table)
```

```
+----------------------+---------------------+---------------+-------------+
|      Vectorizer      |        Model        | Test Log Loss | Improvement |
+----------------------+---------------------+---------------+-------------+
|         NAN          |        Random       |      2.48     |      0      |
| Gene Feature One-Hot |      Linear SVM     |      1.19     |     0.01    |
| Var Feature One-Hot  |      Linear SVM     |      1.73     |     0.0     |
| Text Feature One-Hot |      Linear SVM     |      1.1      |     0.09    |
|  TfIdf All Features  |     Naive Bayes     |      1.17     |     0.1     |
| All feature One-Hot  |         kNN         |      0.98     |     0.12    |
| All feature One-Hot  |       Logistic      |      0.97     |     0.24    |
| All feature One-Hot  |     Logistic Reg    |      1.01     |     0.24    |
| All feature One-Hot  |         kNN         |      1.06     |     0.18    |
| All feature One-Hot  |    Random Forests   |      1.17     |     0.0     |
|  All feat Res Encod  |    Random Forests   |      1.25     |     0.16    |
|   All feat One-hot   | Stacking Classifier |      1.14     |     0.01    |
|   All feat One Hot   |      Max Voting     |      1.19     |     0.03    |
|   All feat One Hot   |     Logistic Reg    |      0.98     |     0.12    |
+----------------------+---------------------+---------------+-------------+
```

Conclusions

In [0]:
```
# So overall in the improvement column we see that replacing the CountVEctorizer
# with TfIdfVectorizer is better as we see a good improvement in the log -loss
# for all different types of models.
# Also, feature engineering techniques are very important here like using max_features,
# min_df,stopwords removal
```