Objective :-

1. Trying the feature engineering techniques to reduce the CV and test log-loss.

Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

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 [0]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

In [2]:
```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages
(1.5.2)
Requirement already satisfied: urllib3<1.23.0,>=1.15 in /usr/local/lib/python3.
6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packa
ges (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-package
s (from kaggle) (2018.11.29)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist
-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packag
es (from kaggle) (2.18.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
(from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/dist-
packages (from kaggle) (2.0.1)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->kaggle) (2.6)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.
6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: Unidecode>=0.04.16 in /usr/local/lib/python3.6/d
ist-packages (from python-slugify->kaggle) (1.0.23)
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[2]:
```
{'kaggle.json': b'{"username":"pankajkarki","key":"6df9de76323f35cb7449790489c9
d534"}'}
```

In [3]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json

!kaggle competitions download -c msk-redefining-cancer-treatment

!ls
```

```
Downloading training_variants.zip to /content
  0% 0.00/24.2k [00:00<?, ?B/s]
100% 24.2k/24.2k [00:00<00:00, 9.35MB/s]
Downloading test_variants.zip to /content
  0% 0.00/47.5k [00:00<?, ?B/s]
100% 47.5k/47.5k [00:00<00:00, 51.7MB/s]
Downloading training_text.zip to /content
 80% 49.0M/61.0M [00:01<00:00, 22.5MB/s]
100% 61.0M/61.0M [00:01<00:00, 39.4MB/s]
Downloading test_text.zip to /content
 92% 91.0M/99.0M [00:00<00:00, 81.3MB/s]
100% 99.0M/99.0M [00:00<00:00, 107MB/s]
Downloading stage2_sample_submission.csv.7z to /content
  0% 0.00/765 [00:00<?, ?B/s]
100% 765/765 [00:00<00:00, 758kB/s]
Downloading stage2_test_variants.csv.7z to /content
  0% 0.00/7.25k [00:00<?, ?B/s]
100% 7.25k/7.25k [00:00<00:00, 7.08MB/s]
Downloading stage2_test_text.csv.7z to /content
  0% 0.00/8.88M [00:00<?, ?B/s]
100% 8.88M/8.88M [00:00<00:00, 81.2MB/s]
Downloading stage1_solution_filtered.csv.7z to /content
  0% 0.00/1.28k [00:00<?, ?B/s]
100% 1.28k/1.28k [00:00<00:00, 1.32MB/s]
Downloading stage_2_private_solution.csv.7z to /content
  0% 0.00/592 [00:00<?, ?B/s]
100% 592/592 [00:00<00:00, 589kB/s]
kaggle.json                        stage2_test_variants.csv.7z
sample_data                        test_text.zip
stage1_solution_filtered.csv.7z    test_variants.zip
stage_2_private_solution.csv.7z    training_text.zip
stage2_sample_submission.csv.7z    training_variants.zip
stage2_test_text.csv.7z
```

In [4]:
```
!unzip test_text.zip
!unzip test_variants.zip
!unzip training_text.zip
!unzip training_variants.zip
```

```
Archive:  test_text.zip
  inflating: test_text
Archive:  test_variants.zip
  inflating: test_variants
Archive:  training_text.zip
  inflating: training_text
Archive:  training_variants.zip
  inflating: training_variants
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [5]:
```
data = pd.read_csv('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[5]:

	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 [6]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TE
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[6]:

	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 [13]:
```
import nltk
nltk.download('stopwords')
# 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
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [14]:
```
#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, "second
```

```
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 : 371.128525 seconds
```

In [52]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[52]:

	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 [53]:
```
result[result.isnull().any(axis=1)]
```

Out[53]:

	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['Variatio
```

In [55]:
```
result[result['ID']==1109]
```

Out[55]:

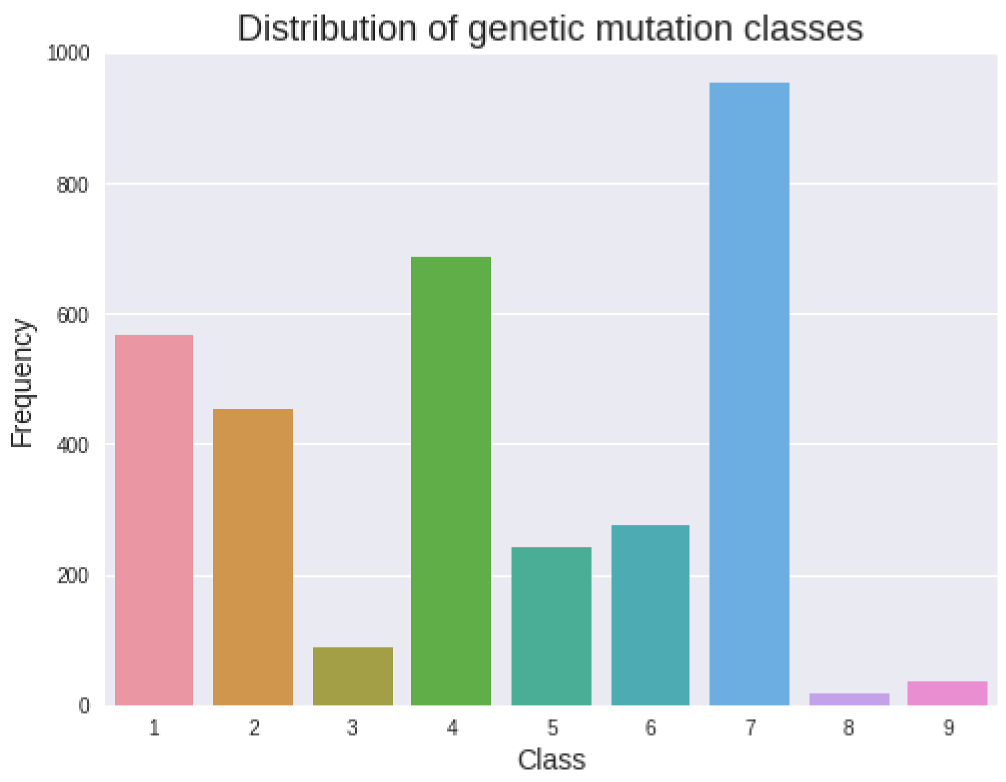
	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

Exploratory Data Analysis

In [57]:
```
data = result
print("For training data, there are a total of", len(data.ID.unique()), "IDs,")
print(len(data.Gene.unique()), "unique genes,")
print(len(data.Variation.unique()), "unique variations and ")
print(len(data.Class.unique()),  "classes")
```

```
For training data, there are a total of 3321 IDs,
264 unique genes,
2996 unique variations and
9 classes
```

In [58]:
```
plt.figure(figsize=(8,6))
sns.countplot(x="Class", data=data)
plt.ylabel('Frequency', fontsize=14)
plt.xlabel('Class', fontsize=14)
plt.title("Distribution of genetic mutation classes", fontsize=18)
plt.show()
```

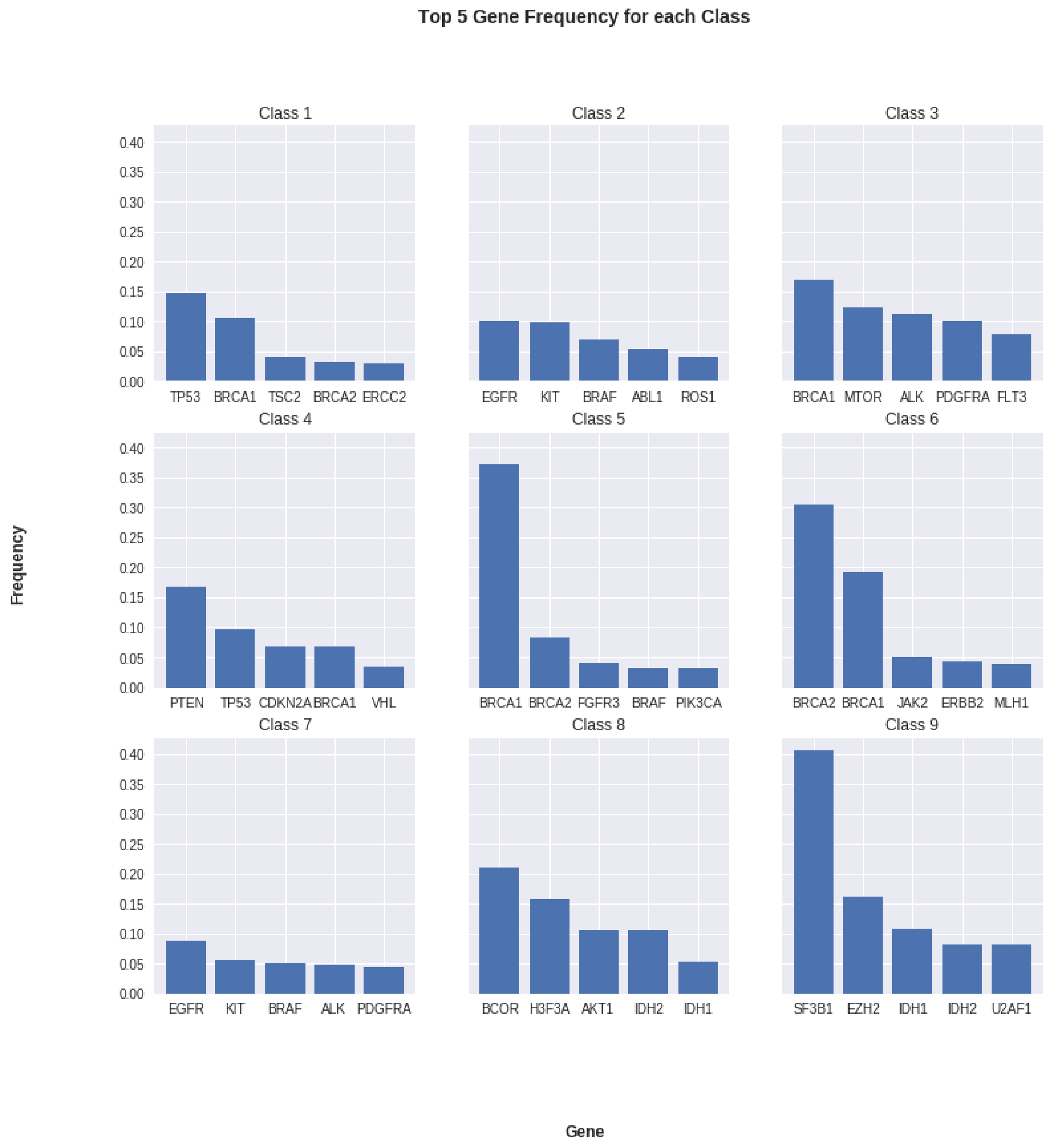


In [59]:
```
train_genes = data.groupby("Gene")['Gene'].count()
fewest_genes = train_genes.sort_values(ascending=True)[:10]
print("Genes with most occurences\n", train_genes.sort_values(ascending=False)[:1
print("\nGenes with fewest occurences\n", fewest_genes)
```

```
Genes with most occurences
 Gene
BRCA1     264
TP53      163
EGFR      141
PTEN      126
BRCA2     125
KIT        99
BRAF       93
ERBB2      69
ALK        69
PDGFRA     60
Name: Gene, dtype: int64

Genes with fewest occurences
 Gene
KLF4      1
FGF19     1
FANCC     1
FAM58A    1
PAK1      1
ERRFI1    1
PAX8      1
PIK3R3    1
PMS1      1
PPM1D     1
Name: Gene, dtype: int64
```

In [60]:
```
# Genes with highest frequencies in each class
fig, axes = plt.subplots(nrows=3, ncols=3, sharey=True, figsize=(12,12))

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

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

fig.text(0.5, 0.97, 'Top 5 Gene Frequency for each Class', ha='center', fontsize=
fig.text(0.5, 0, 'Gene', ha='center', fontweight='bold')
fig.text(0, 0.5, 'Frequency', va='center', rotation='vertical', fontweight='bold'
```

Out[60]:
```
Text(0, 0.5, 'Frequency')
```

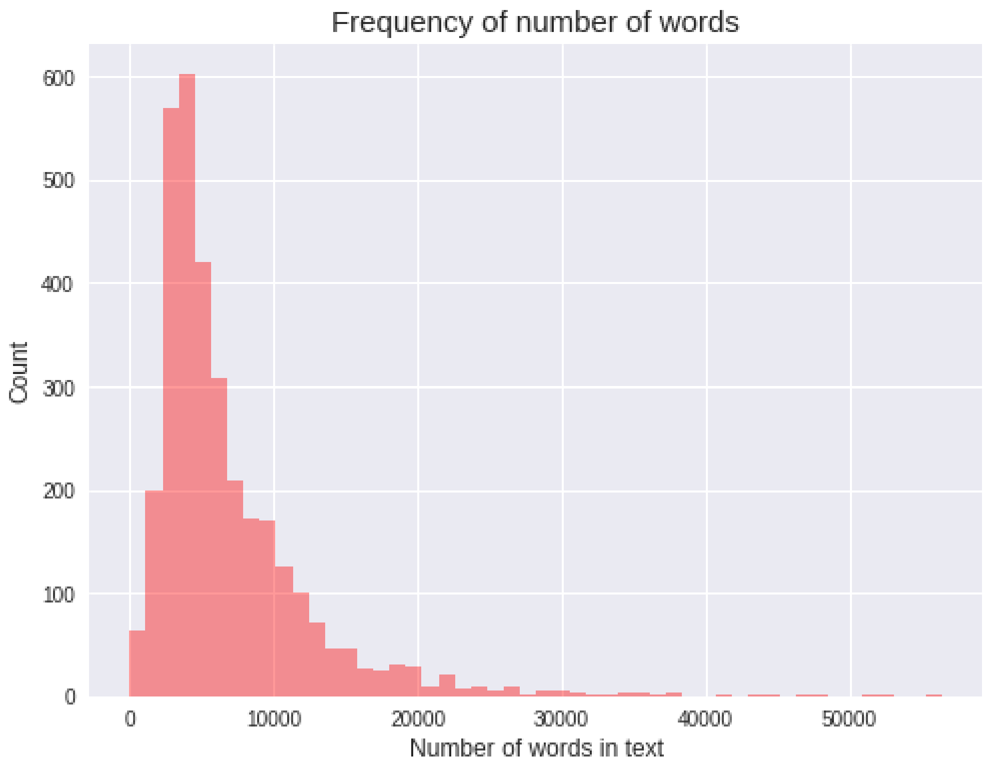


In [61]:
```
data['Text_count']  = data["TEXT"].apply(lambda x: len(str(x).split()))
data.head()
```

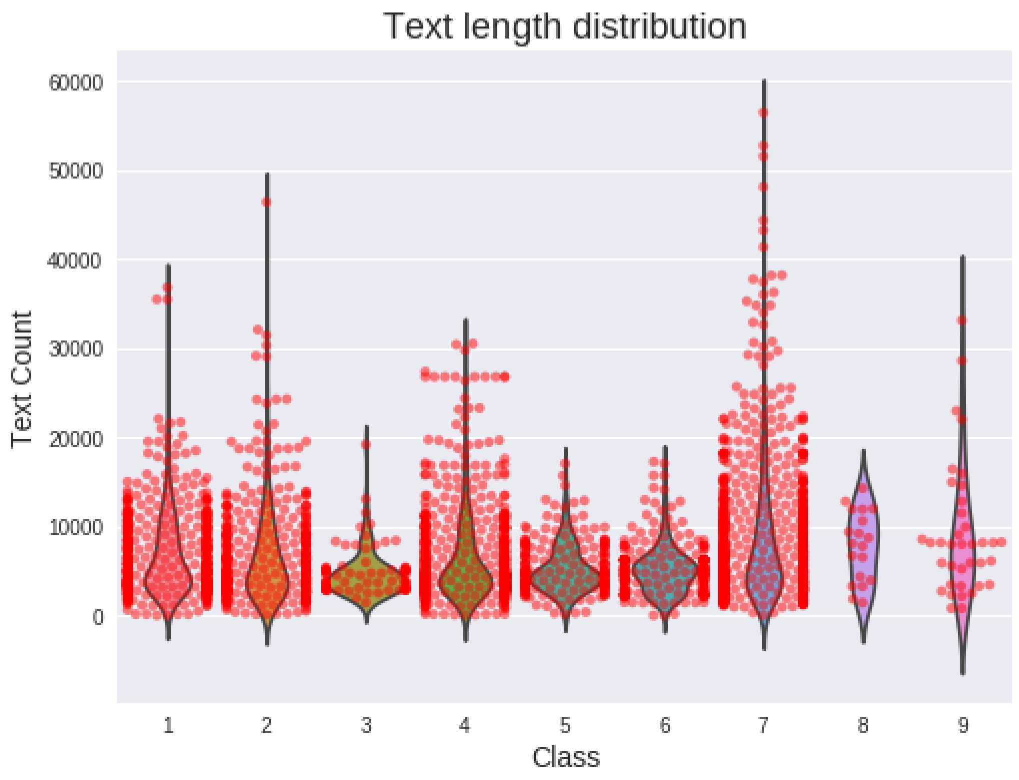
Out[61]:

	ID	Gene	Variation	Class	TEXT	Text_count
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	4370
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	4139
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	4139
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	3841
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	4254

In [62]:
```
#Frequency of no. of words in text
plt.figure(figsize=(8, 6))
sns.distplot(data.Text_count.values, bins=50, kde=False, color='r')
plt.xlabel('Number of words in text', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title("Frequency of number of words", fontsize=15)
plt.show()
```



In [63]:
```
# Distribution of text count for each class - violin plot
plt.figure(figsize=(8,6))
gene_count_grp = data.groupby('Gene')["Text_count"].sum().reset_index()
sns.violinplot(x="Class", y="Text_count", data=data, inner=None)
sns.swarmplot(x="Class", y="Text_count", data=data, color="r", alpha=.5);
plt.ylabel('Text Count', fontsize=14)
plt.xlabel('Class', fontsize=14)
plt.title("Text length distribution", fontsize=18)
plt.show()
```



Observation

1. In first plot, 3rd, 8th and 9th Class have very less datapoints because of which data is imbalanced.
2. In second plot, We can see that the most frequent genes in each classes are very different.
3. In third plot, We can see that all the classes have text counts between 0-20,000

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 v
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_t
# split the train data into train and cross validation by maintaining same distri
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_tr
```

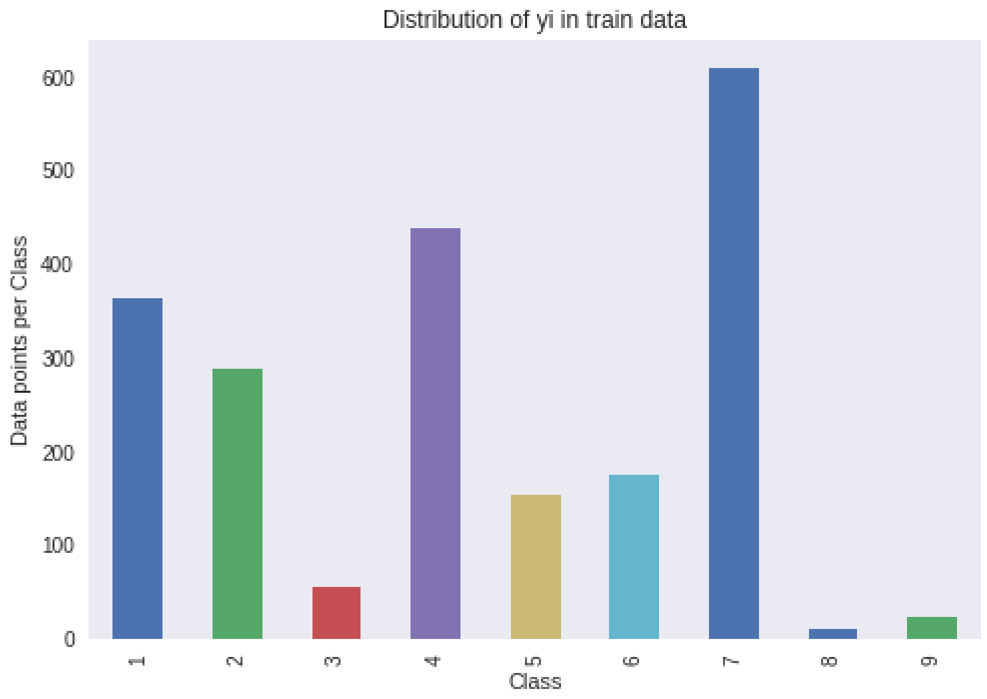
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [16]:
```
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])
```

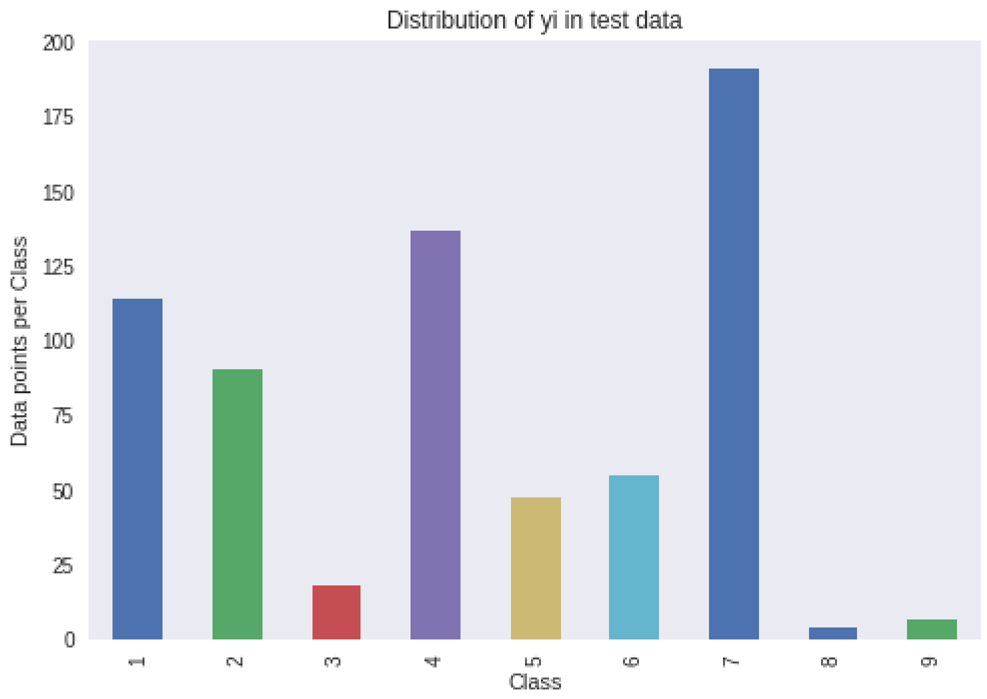
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

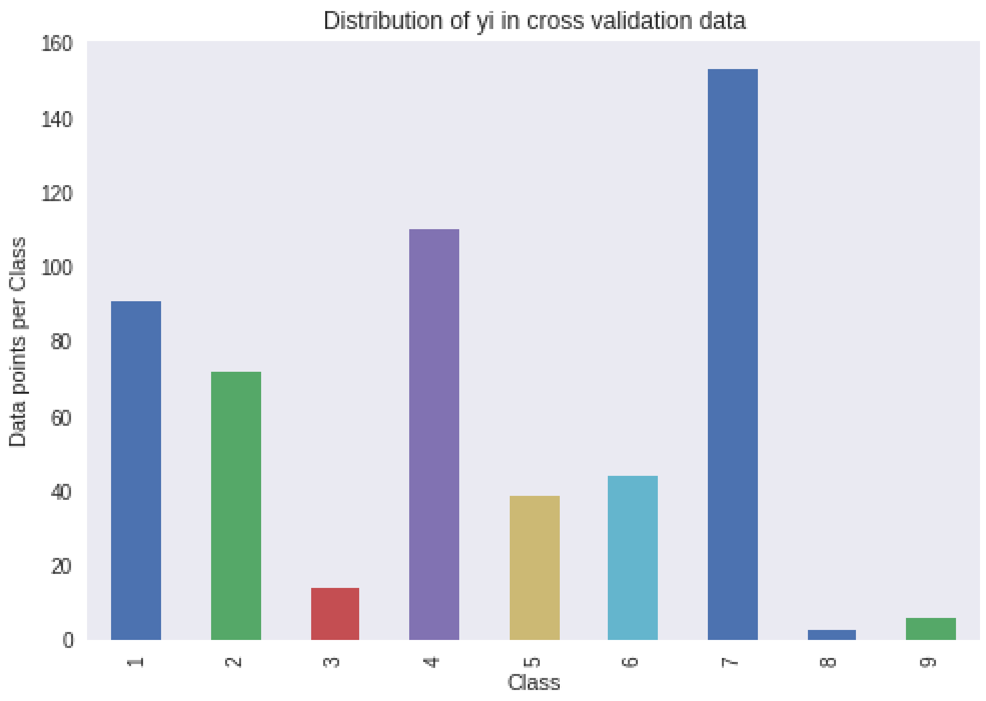
In [17]:
```
# it returns a dict, keys as class labels and values as the number of data points
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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.val


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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.valu

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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
```



```
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 %)
```

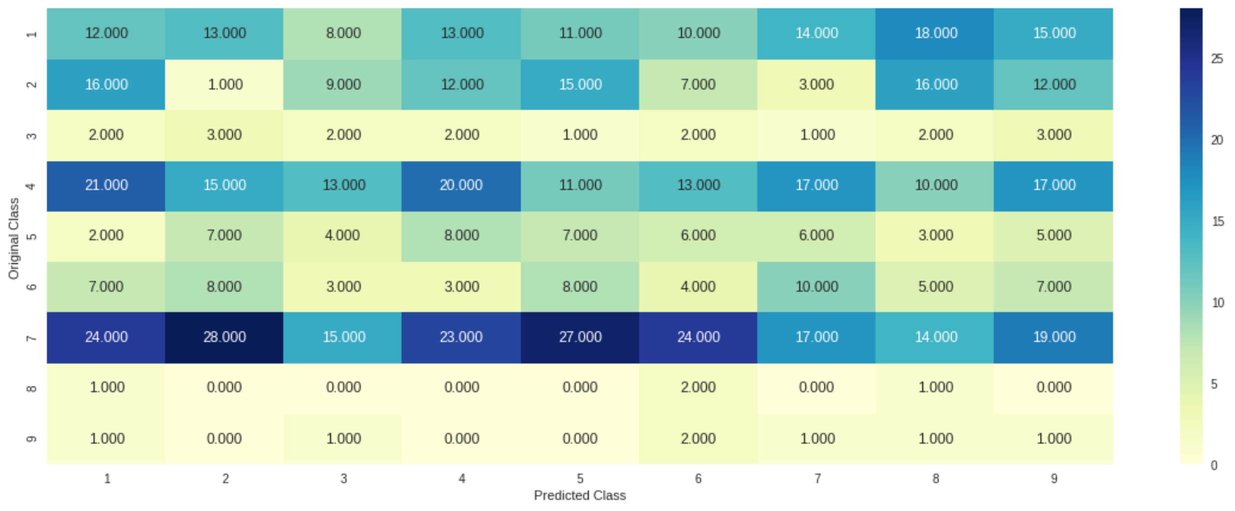
3.2 Prediction using a 'Random' Model

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 

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

    # 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 row
    # 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 
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to row
    # 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, ytic
    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, ytic
    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, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

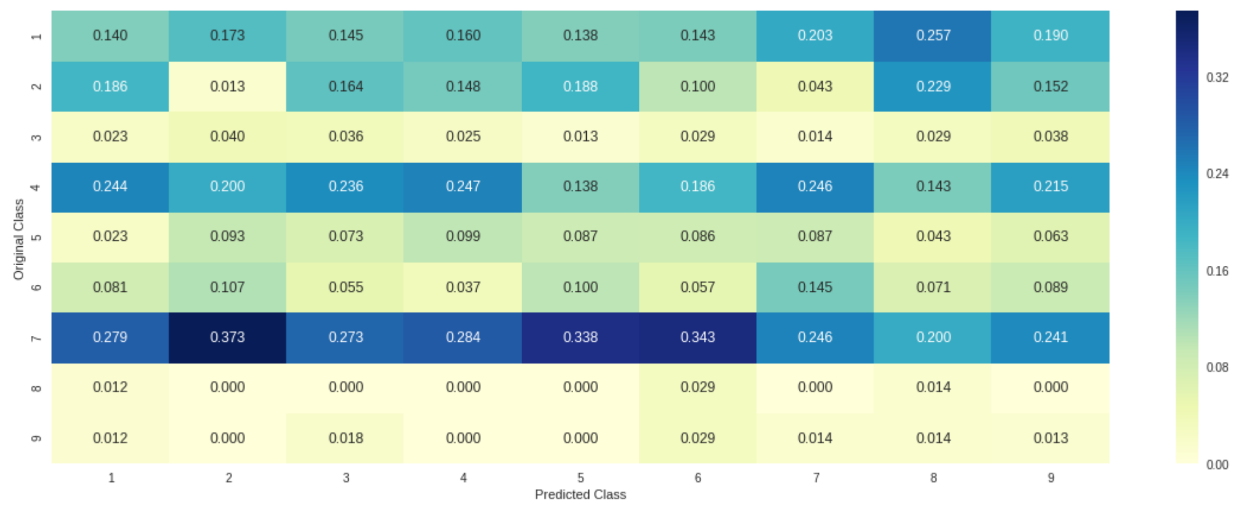
In [19]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their su
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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_pre


# 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

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 2.519642065472273
Log loss on Test Data using Random Model 2.4619529686123123
-------------------- Confusion matrix --------------------
```

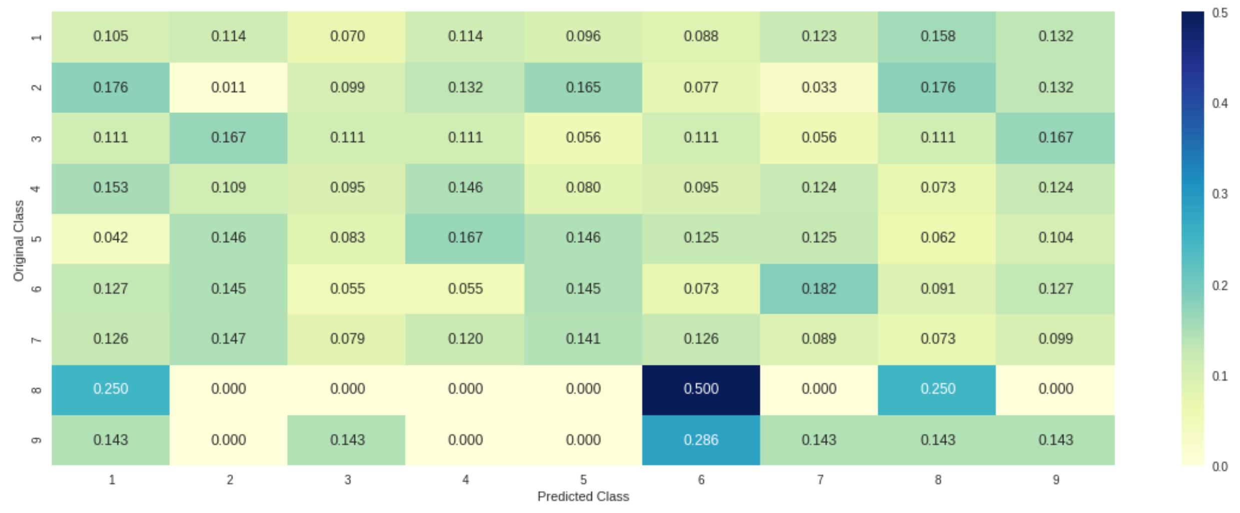


	1	2	3	4	5	6	7	8	9
1	12.000	13.000	8.000	13.000	11.000	10.000	14.000	18.000	15.000
2	16.000	1.000	9.000	12.000	15.000	7.000	3.000	16.000	12.000
3	2.000	3.000	2.000	2.000	1.000	2.000	1.000	2.000	3.000
4	21.000	15.000	13.000	20.000	11.000	13.000	17.000	10.000	17.000
5	2.000	7.000	4.000	8.000	7.000	6.000	6.000	3.000	5.000
6	7.000	8.000	3.000	3.000	8.000	4.000	10.000	5.000	7.000
7	24.000	28.000	15.000	23.000	27.000	24.000	17.000	14.000	19.000
8	1.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000	0.000
9	1.000	0.000	1.000	0.000	0.000	2.000	1.000	1.000	1.000

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



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



3.3 Univariate Analysis

In [0]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in tra
# build a vector (1*9) , the first element = (number of times it occured in class
# gv_dict is like a look up table, for every gene it store a (1*9) representation
# 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()
    #print(value_count)

    # gv_dict : Gene Variation Dict, which contains the probability array for eac
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRC
            #         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 pa
            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
        #print(gv_dict)
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818181
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897959183
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060606060606060
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918238993710
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920529
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333333333333333
    #      ...
    #     }
    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 featur
    gv_fea = []
    # for every feature values in the given data frame we will check if it is the
    # 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?

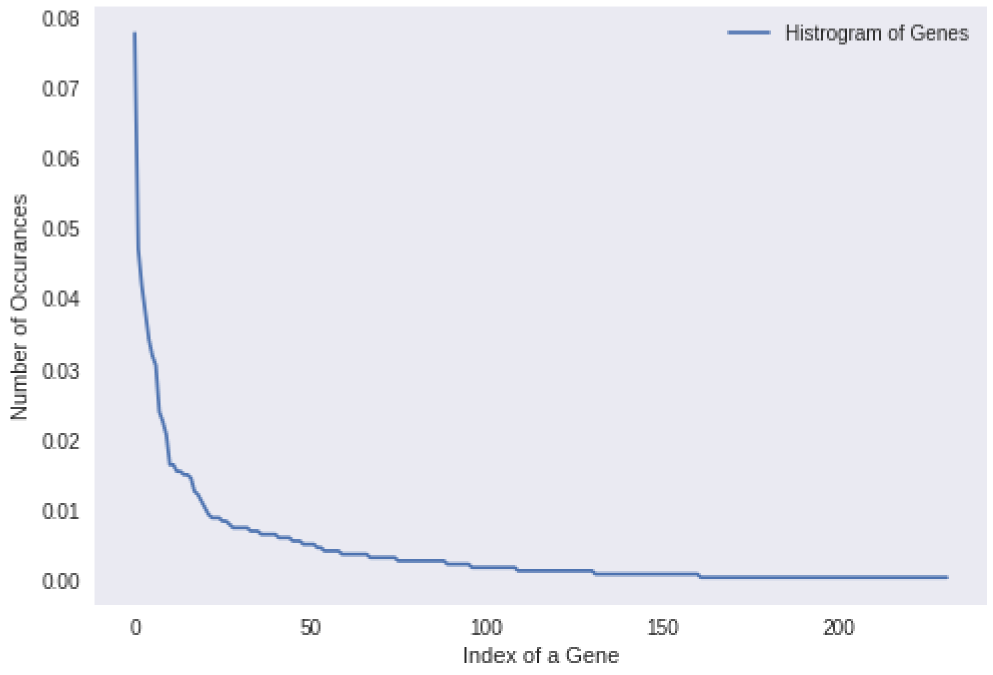
In [21]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1     165
TP53      100
EGFR       89
PTEN       81
BRCA2      73
BRAF       68
KIT        65
ALK        51
ERBB2      48
PDGFRA     44
Name: Gene, dtype: int64
```

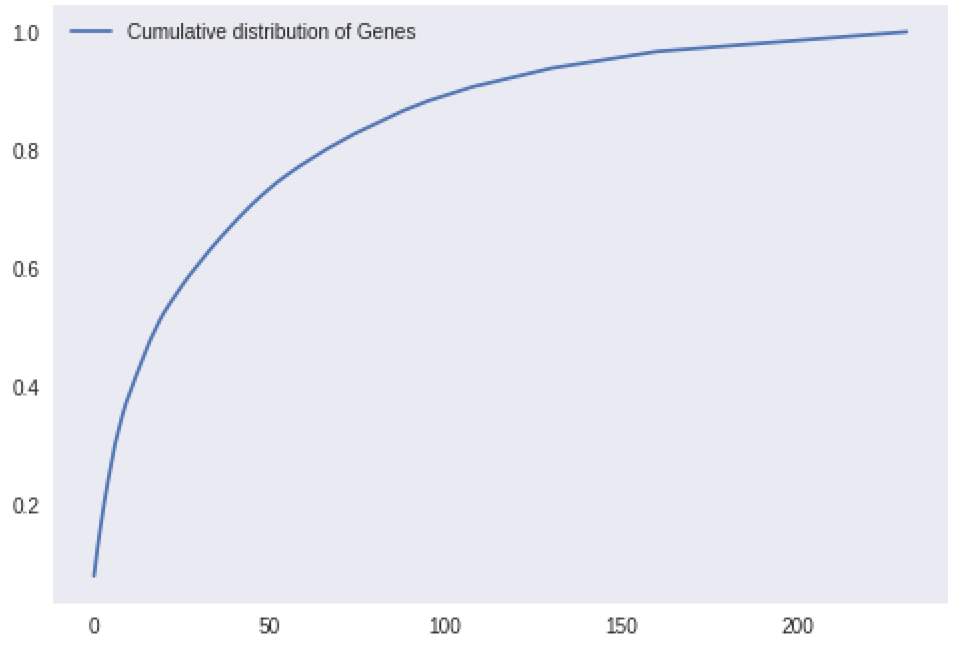
In [22]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in 
```

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

In [23]:
```
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 [24]:
```
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/ (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_
# 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 codin
```

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

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

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

Out[283]:
```
1412     FGFR3
21         CBL
504       TP53
2126     CCND1
884     PDGFRA
Name: Gene, dtype: object
```

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

Out[284]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
```

In [0]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encodin
```

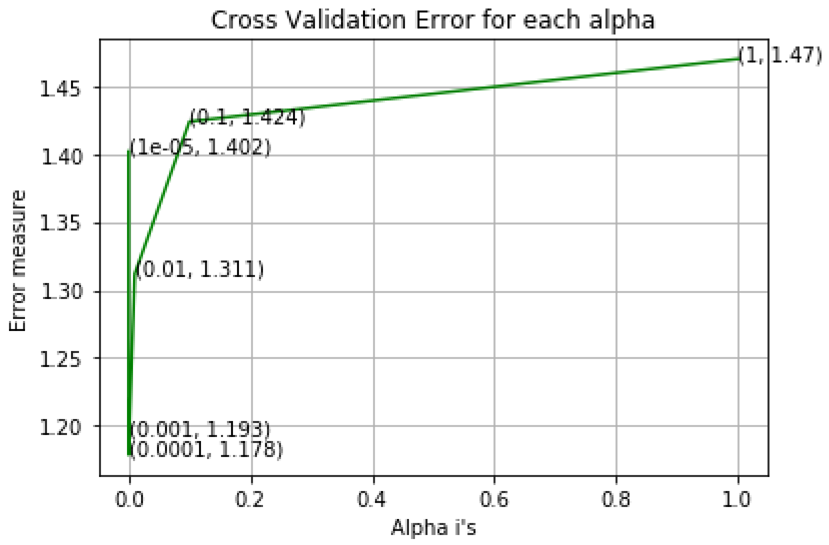
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding met
hod. The shape of gene feature: (2124, 242)
```

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/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_sta
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:",
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
For values of alpha =  1e-05 The log loss is: 1.4019625605579298
For values of alpha =  0.0001 The log loss is: 1.178390947518176
For values of alpha =  0.001 The log loss is: 1.1932544784269383
For values of alpha =  0.01 The log loss is: 1.3114440889211676
For values of alpha =  0.1 The log loss is: 1.424005362413235
For values of alpha =  1 The log loss is: 1.4701532358158753
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0604543628136502
For values of best alpha =  0.0001 The cross validation log loss is: 1.17839094
7518176
For values of best alpha =  0.0001 The test log loss is: 1.192214262109507
```

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 ", uni

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_
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(c
```

```
Q6. How many data points in Test and CV datasets are covered by the  242  genes
in train dataset?
Ans
1. In test data 655 out of 665 : 98.49624060150376
2. In cross validation data 518 out of  532 : 97.36842105263158
```

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?

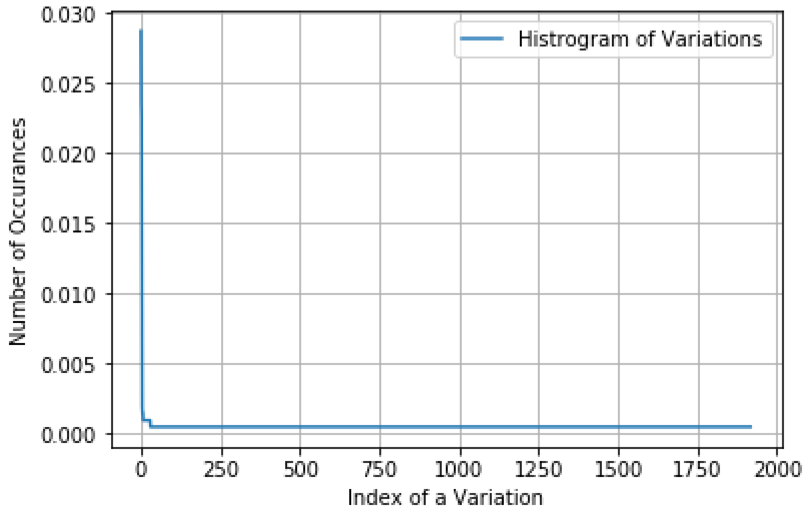
```
In [0]: unique_variations = train_df['Variation'].value_counts()
        print('Number of Unique Variations :', unique_variations.shape[0])
        # the top 10 variations that occured most
        print(unique_variations.head(10))
```

```
Number of Unique Variations : 1918
Truncating_Mutations    61
Amplification           50
Deletion                48
Fusions                 21
Overexpression           4
Q61H                     3
Q61R                     3
G12V                     2
G35R                     2
E330K                    2
Name: Variation, dtype: int64
```

```
In [0]: print("Ans: There are", unique_variations.shape[0] ,"different categories of vari
```

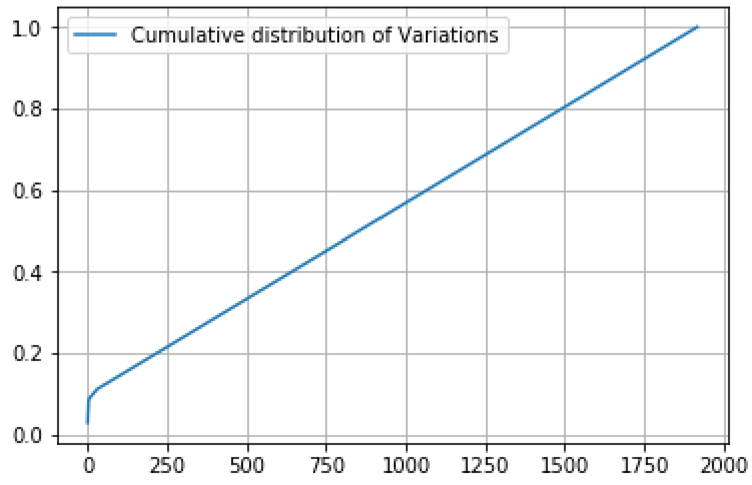
```
Ans: There are 1918 different categories of variations in the train data, and t
hey are distibuted as follows
```

```
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()
```

```
[0.0287194  0.05225989 0.07485876 ... 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/ (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, "Variatio
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",
```

In [0]:
```
print("train_variation_feature_responseCoding is a converted feature using the re
```

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

In [0]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_d
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Var
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variati
```

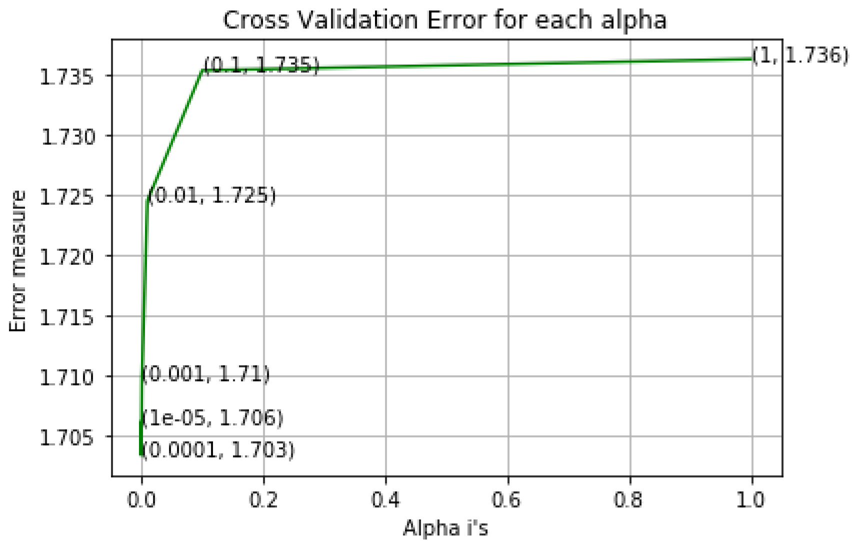
In [0]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot e
ncoding method. The shape of Variation feature: (2124, 1947)
```

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/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_sta
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:",
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
For values of alpha =  1e-05 The log loss is: 1.7061033297245032
For values of alpha =  0.0001 The log loss is: 1.7033279641939136
For values of alpha =  0.001 The log loss is: 1.7096735602184494
For values of alpha =  0.01 The log loss is: 1.7245401724431388
For values of alpha =  0.1 The log loss is: 1.735425941378018
For values of alpha =  1 The log loss is: 1.7363709412512816
```



```
For values of best alpha =  0.0001 The train log loss is: 0.799892940719126
For values of best alpha =  0.0001 The cross validation log loss is: 1.70332796
41939136
For values of best alpha =  0.0001 The test log loss is: 1.7045757654046538
```

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

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

In [0]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shap
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(c
```

```
Q12. How many data points are covered by total  1918  genes in test and cross v
alidation data sets?
Ans
1. In test data 62 out of 665 : 9.323308270676693
2. In cross validation data 54 out of  532 : 10.150375939849624
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?

5. Is the text feature stable across train, test and CV datasets?

In [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.
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row
            row_index += 1
    return text_feature_responseCoding
```

In [35]:
```
# building a TfidfVectorizer with all the words that occured minimum 3 times in tı
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 (.
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 tim
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_te
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_featur
```

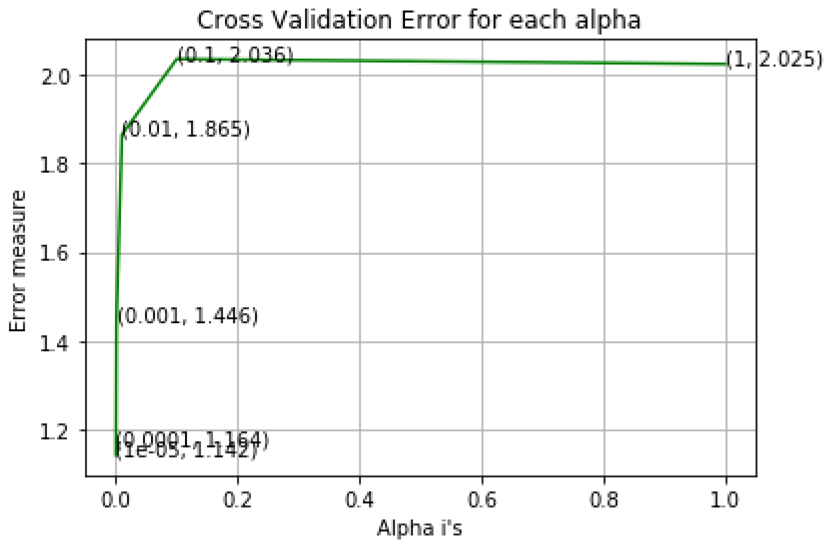
In [0]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis

# 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] , re
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

```
In [0]: # Number of words for a given frequency.
        print(Counter(sorted_text_occur))
```

```
Counter({254.92485564185105: 1, 179.6743366077026: 1, 138.48643727952754: 1,
129.80535245216572: 1, 128.96575874866505: 1, 119.7414576102236: 1, 119.3522
720822652: 1, 115.01476649092555: 1, 111.76421625142224: 1, 109.561742518618
37: 1, 105.76516641704632: 1, 90.55426332692564: 1, 89.52704545646593: 1, 8
8.9021199890866: 1, 84.94280731801689: 1, 81.66471897938335: 1, 80.553541047
50046: 1, 80.18581330223554: 1, 79.45409353776236: 1, 75.39402760248896: 1,
75.11643642699688: 1, 75.08425737095087: 1, 71.05099875888212: 1, 68.8517708
320724: 1, 68.74832287501928: 1, 67.4351502441337: 1, 66.2649713448625: 1, 6
6.1604432175213: 1, 64.64542410454514: 1, 64.516623063368: 1, 64.43174995244
321: 1, 63.519629816470804: 1, 62.940623589195624: 1, 60.48918250791442: 1,
59.32698200783694: 1, 57.511612143086126: 1, 56.321345852853554: 1, 56.21697
3923270736: 1, 54.552294218017856: 1, 54.439117161411474: 1, 50.707308096549
46: 1, 49.557820346795154: 1, 49.383113608295645: 1, 49.001611717154674: 1,
48.42991228180798: 1, 48.23485103488938: 1, 48.226458726564466: 1, 46.346273
59000825: 1, 46.262150699962646: 1, 45.028466083698: 1, 44.50598663581978:
1, 43.57949679729767: 1, 43.52860726352477: 1, 43.47404694295213: 1, 43.4245
15813504996: 1, 43.17842428586127: 1, 42.76092293218395: 1, 42.6814979866371
4: 1, 42.33624327888522: 1, 42.17326616916091: 1, 42.124955794747514: 1, 41.
62745353422218: 1, 41.41720665335359: 1, 41.0906331938435: 1, 40.99655066806
```

In [0]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_sta
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:",
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
For values of alpha =  1e-05 The log loss is: 1.1417877137308243
For values of alpha =  0.0001 The log loss is: 1.1637328605378794
For values of alpha =  0.001 The log loss is: 1.4457432306630014
For values of alpha =  0.01 The log loss is: 1.8651768087409692
For values of alpha =  0.1 The log loss is: 2.03622483427236
For values of alpha =  1 The log loss is: 2.024942550444193
```



```
For values of best alpha =  1e-05 The train log loss is: 0.8079914187375853
For values of best alpha =  1e-05 The cross validation log loss is: 1.141787713
7308243
For values of best alpha =  1e-05 The test log loss is: 1.1500729891599621
```

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

Ans. Yes, it seems like!

In [0]:
```python
def get_intersec_text(df):
    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]:
```python
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train dat
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in tr
```

```
94.7 % of word of test data appeared in train data
94.0 % 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 belong
    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))
    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 = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    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 [{}]".form
        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 [{}]"
        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 [{}]".form

    print("Out of the top ",no_features," features ", word_present, "are present
```

Feature engineering

In [0]:
```
# Combine Gene & Variation features
Gene_Var_data=[]
for i in data["Gene"].values:
    Gene_Var_data.append(i)
for i in data["Variation"].values:
    Gene_Var_data.append(i)

# Building TF-IDF vectorizer ontop of combined gene_var_data
cnt_vect=TfidfVectorizer()
Fit_vect=cnt_vect.fit(Gene_Var_data)
train_text_vect=cnt_vect.transform(train_df["TEXT"])
cv_text_vect=cnt_vect.transform(cv_df["TEXT"])
test_text_vect=cnt_vect.transform(test_df["TEXT"])

# Normalization
train_text_vect=normalize(train_text_vect,axis=0)
cv_text_vect=normalize(cv_text_vect,axis=0)
test_text_vect=normalize(test_text_vect,axis=0)
```

Stacking the three types of features

And also I stacking that with one hot encoding 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_varia
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variatio
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feat

train_gene_var_updt=hstack((train_gene_var_onehotCoding,train_text_vect))
train_x_onehotCoding = hstack((train_gene_var_updt, train_text_feature_onehotCodi
train_y = np.array(list(train_df['Class']))


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


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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,trai
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_v
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variati

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_fea
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_featur
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_resp
```

In [0]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_on
print("(number of data points * number of features) in test data = ", test_x_oneh
print("(number of data points * number of features) in cross validation data =",
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 6427)
(number of data points * number of features) in test data =  (665, 6427)
(number of data points * number of features) in cross validation data = (532, 6
427)
```

In [0]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_re
print("(number of data points * number of features) in test data = ", test_x_resp
print("(number of data points * number of features) in cross validation data =",
```

```
 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, 2
7)
```

4.1. Base Line Model

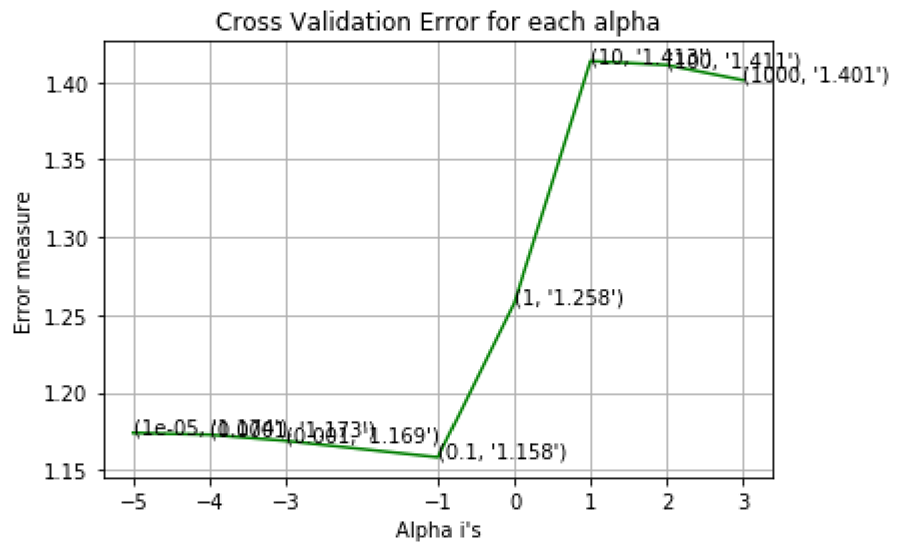
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [0]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/st
# -------------------------
# 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/les
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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/les
# -----------------------


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_,
    # to avoid rounding error while multiplying probabilites we use log-probabili
    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:",
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
for alpha = 1e-05
Log Loss : 1.1739581761584255
for alpha = 0.0001
Log Loss : 1.1728047951914005
for alpha = 0.001
Log Loss : 1.168864539462507
for alpha = 0.1
Log Loss : 1.158250468831457
for alpha = 1
Log Loss : 1.2577399951875554
for alpha = 10
Log Loss : 1.4134772942015252
for alpha = 100
Log Loss : 1.4109445895566923
for alpha = 1000
Log Loss : 1.4012076802171929
```

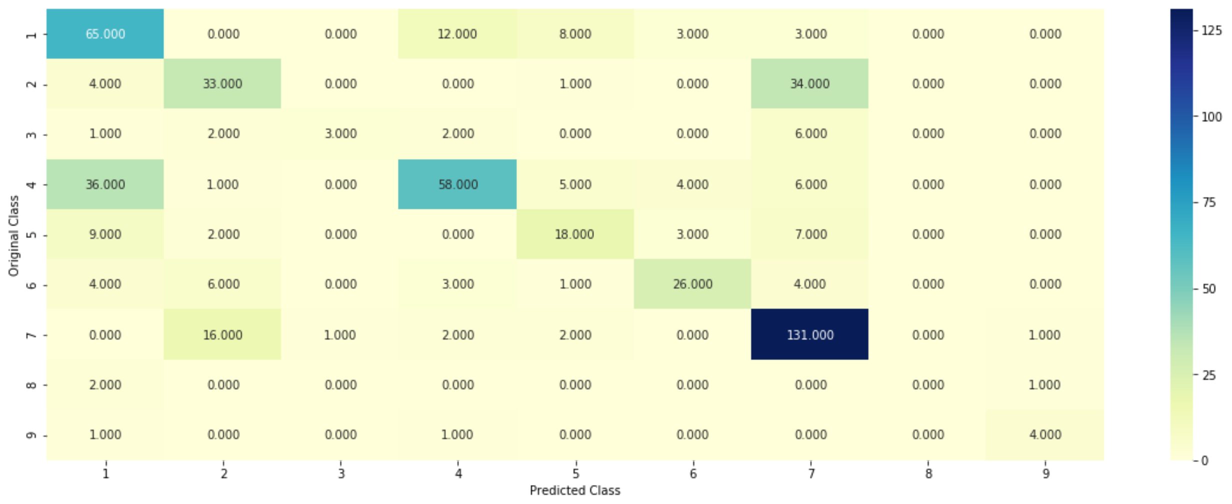


```
For values of best alpha =  0.1 The train log loss is: 0.7606205701319388
For values of best alpha =  0.1 The cross validation log loss is: 1.15825046883
1457
For values of best alpha =  0.1 The test log loss is: 1.1881531434320403
```

4.1.1.2. Testing the model with best hyper paramters

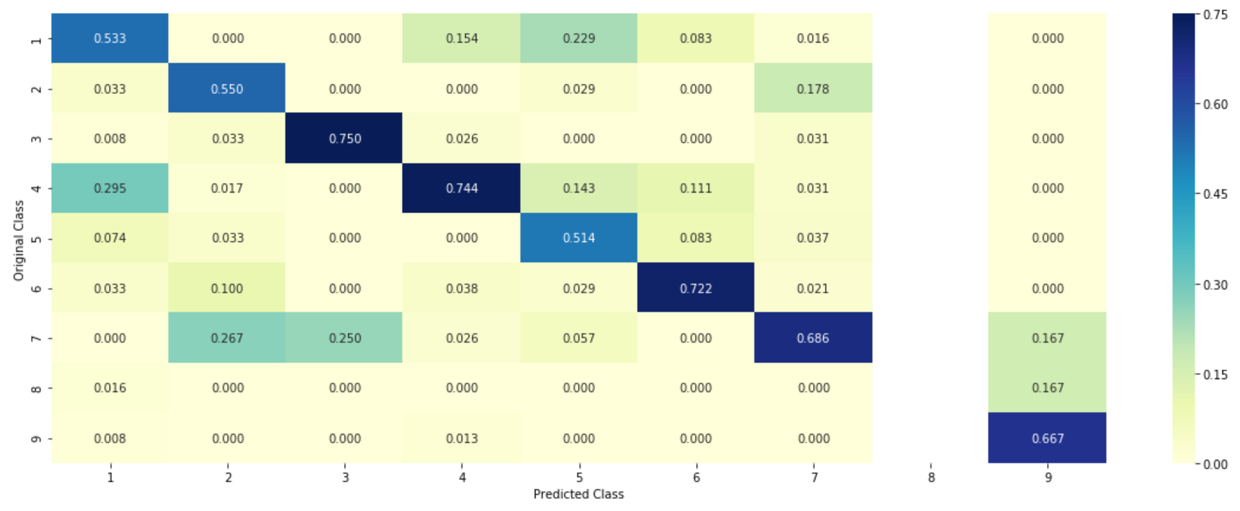
In [0]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/st
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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/les
# -----------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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 e
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.158250468831457
Number of missclassified point : 0.36466165413533835
-------------------- Confusion matrix --------------------
```

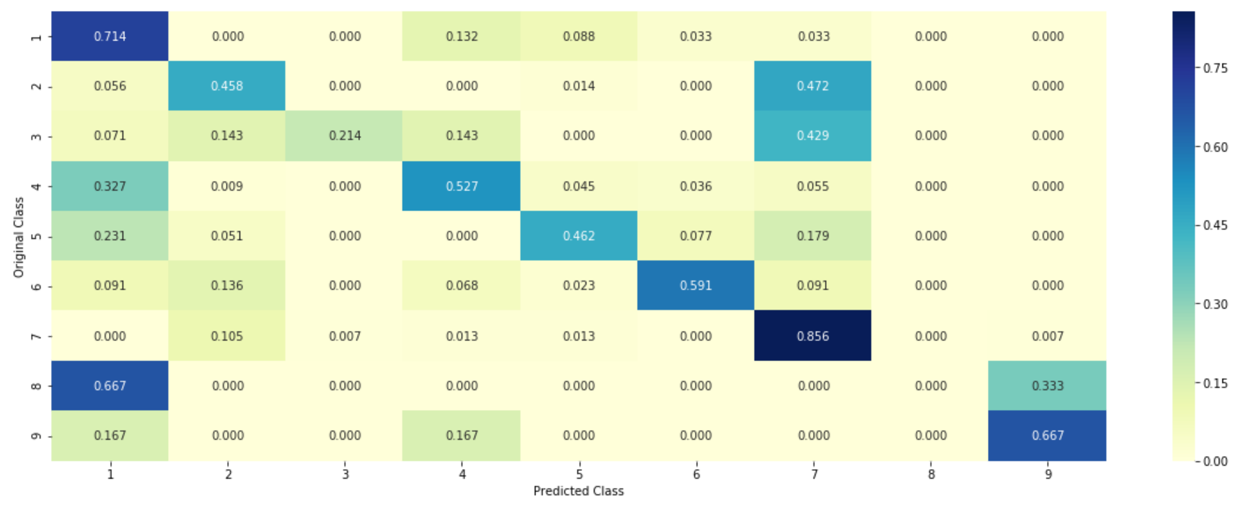


	1	2	3	4	5	6	7	8	9
1	65.000	0.000	0.000	12.000	8.000	3.000	3.000	0.000	0.000
2	4.000	33.000	0.000	0.000	1.000	0.000	34.000	0.000	0.000
3	1.000	2.000	3.000	2.000	0.000	0.000	6.000	0.000	0.000
4	36.000	1.000	0.000	58.000	5.000	4.000	6.000	0.000	0.000
5	9.000	2.000	0.000	0.000	18.000	3.000	7.000	0.000	0.000
6	4.000	6.000	0.000	3.000	1.000	26.000	4.000	0.000	0.000
7	0.000	16.000	1.000	2.000	2.000	0.000	131.000	0.000	1.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	1.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	4.000

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



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



4.1.1.3. Feature Importance, Correctly classified point

In [0]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.056  0.5863 0.0179 0.0976 0.052  0.0435 0.13
46 0.0064 0.0057]]
Actual Class : 2
--------------------------------------------------
```

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_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0472 0.0494 0.0151 0.0824 0.0439 0.0367 0.71
5  0.0054 0.0048]]
Actual Class : 7
--------------------------------------------------
```

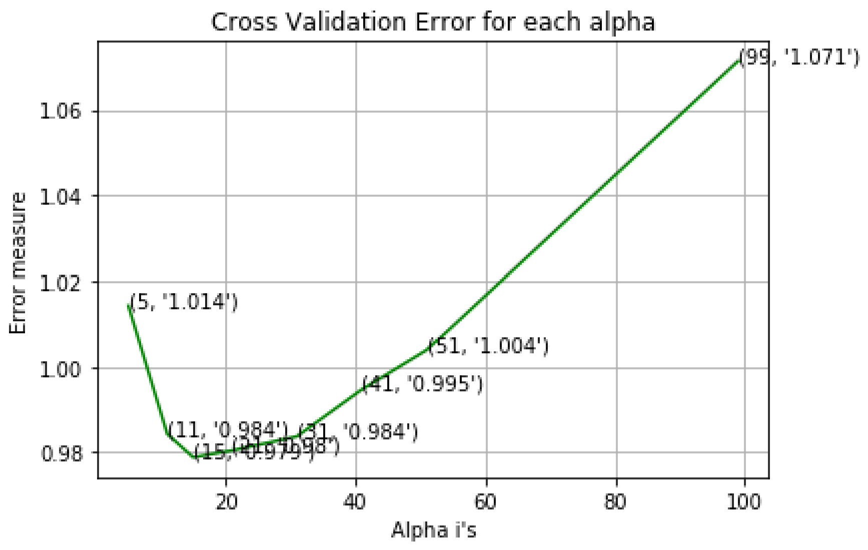
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [0]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# 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/les
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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_,
    # to avoid rounding error while multiplying probabilites we use log-probabili
    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:",
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
for alpha = 5
Log Loss : 1.0142427410600647
for alpha = 11
Log Loss : 0.9843009431025068
for alpha = 15
Log Loss : 0.9788393451387649
for alpha = 21
Log Loss : 0.9804274194543124
for alpha = 31
Log Loss : 0.9837139436419055
for alpha = 41
Log Loss : 0.9946640234837321
for alpha = 51
Log Loss : 1.003891538065869
for alpha = 99
Log Loss : 1.071359160401408
```

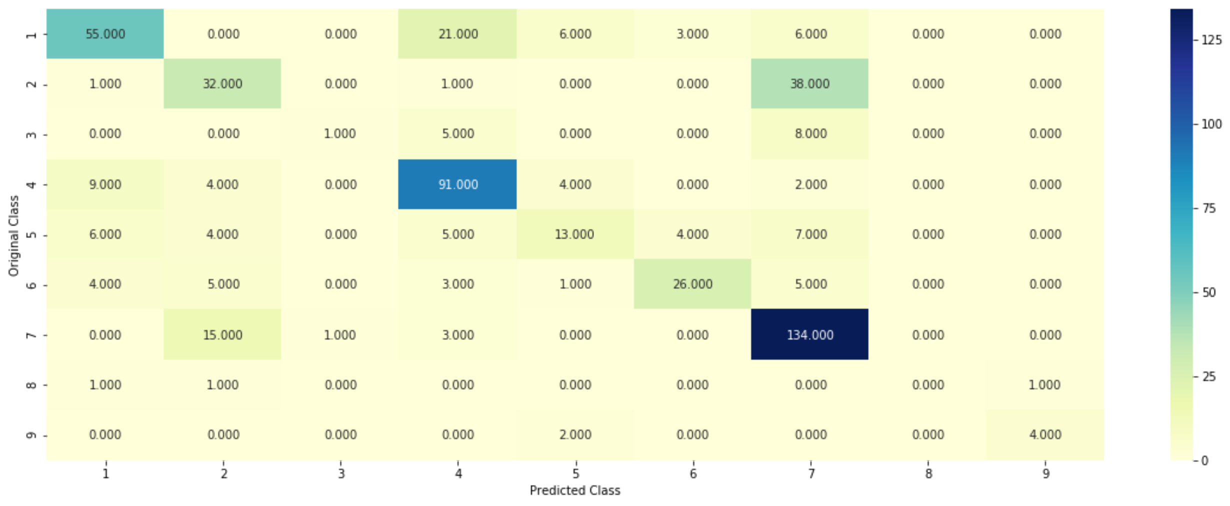


```
For values of best alpha =  15 The train log loss is: 0.7189457900171621
For values of best alpha =  15 The cross validation log loss is: 0.978839345138
7649
For values of best alpha =  15 The test log loss is: 1.055113934016944
```

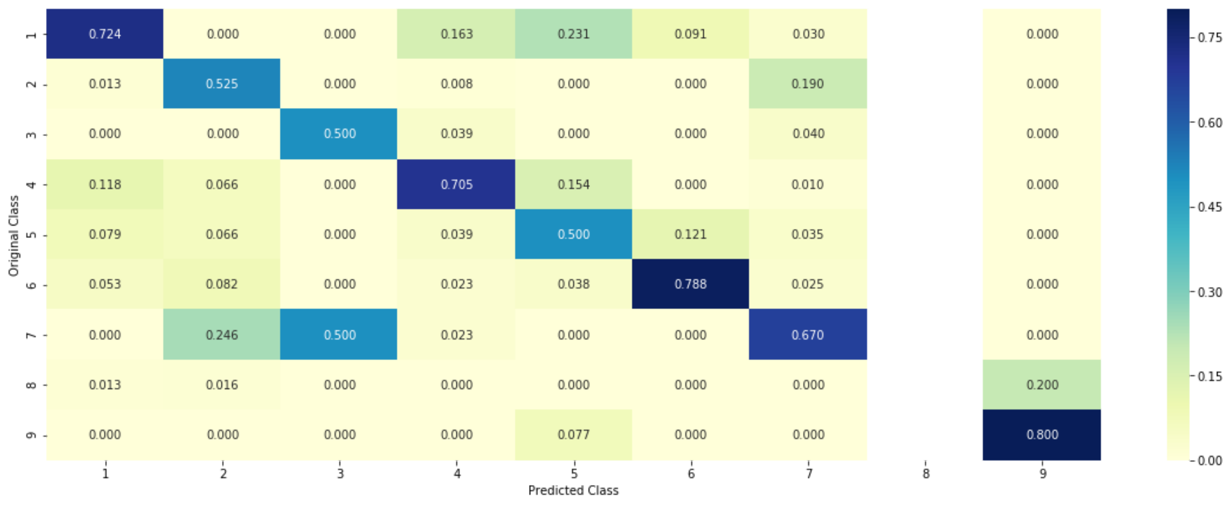
4.2.2. Testing the model with best hyper paramters

In [0]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# 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/les
#------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseC
```

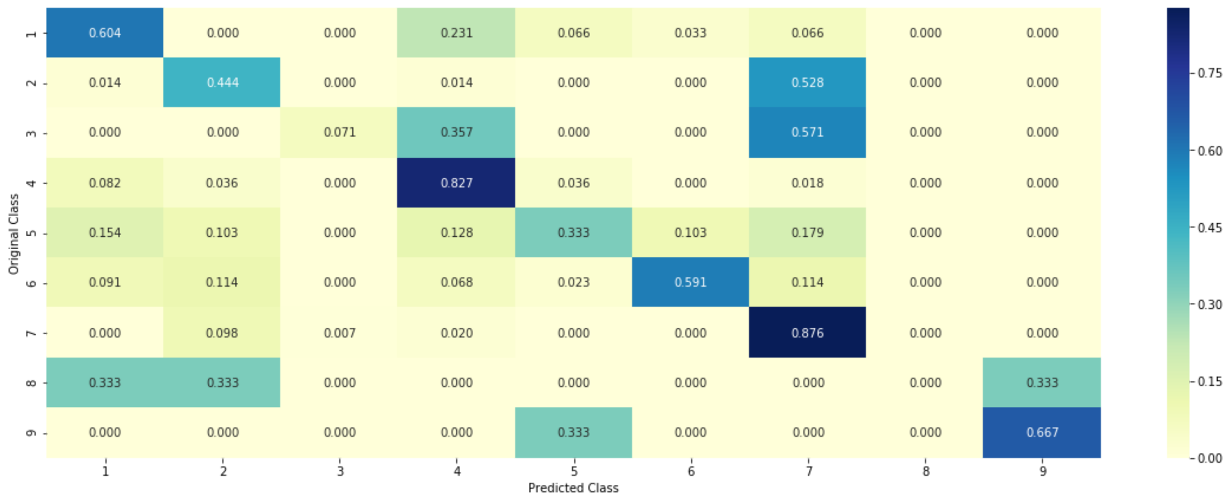
```
Log loss : 0.9788393451387649
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.604	0.000	0.000	0.231	0.066	0.033	0.066	0.000	0.000
2	0.014	0.444	0.000	0.014	0.000	0.000	0.528	0.000	0.000
3	0.000	0.000	0.071	0.357	0.000	0.000	0.571	0.000	0.000
4	0.082	0.036	0.000	0.827	0.036	0.000	0.018	0.000	0.000
5	0.154	0.103	0.000	0.128	0.333	0.103	0.179	0.000	0.000
6	0.091	0.114	0.000	0.068	0.023	0.591	0.114	0.000	0.000
7	0.000	0.098	0.007	0.020	0.000	0.000	0.876	0.000	0.000
8	0.333	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.333
9	0.000	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.667

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)
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 2
The  15  nearest neighbours of the test points belongs to classes [2 2 2 2 7 2
2 2 2 2 2 2 2 2 2]
Fequency of nearest points : Counter({2: 14, 7: 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
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)
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of t
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs
to classes [7 7 7 7 7 3 7 7 7 7 7 7 7 7 7]
Fequency of nearest points : Counter({7: 14, 3: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

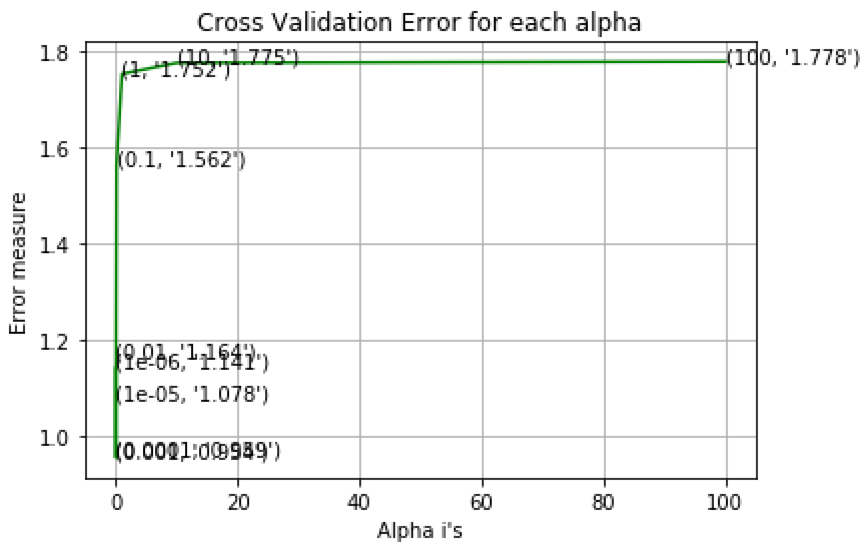
4.3.1.1. Hyper paramter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/les
#------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mo
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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
    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_,
    # to avoid rounding error while multiplying probabilites we use log-probabili
    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
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:",
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
for alpha = 1e-06
Log Loss : 1.1413576067484803
for alpha = 1e-05
Log Loss : 1.0777443009202097
for alpha = 0.0001
Log Loss : 0.9593629043270478
for alpha = 0.001
Log Loss : 0.9541478702481345
for alpha = 0.01
Log Loss : 1.1642179922517195
for alpha = 0.1
Log Loss : 1.5618342052704404
for alpha = 1
Log Loss : 1.752104678422237
for alpha = 10
Log Loss : 1.7749643040101175
for alpha = 100
Log Loss : 1.7777065191600168
```

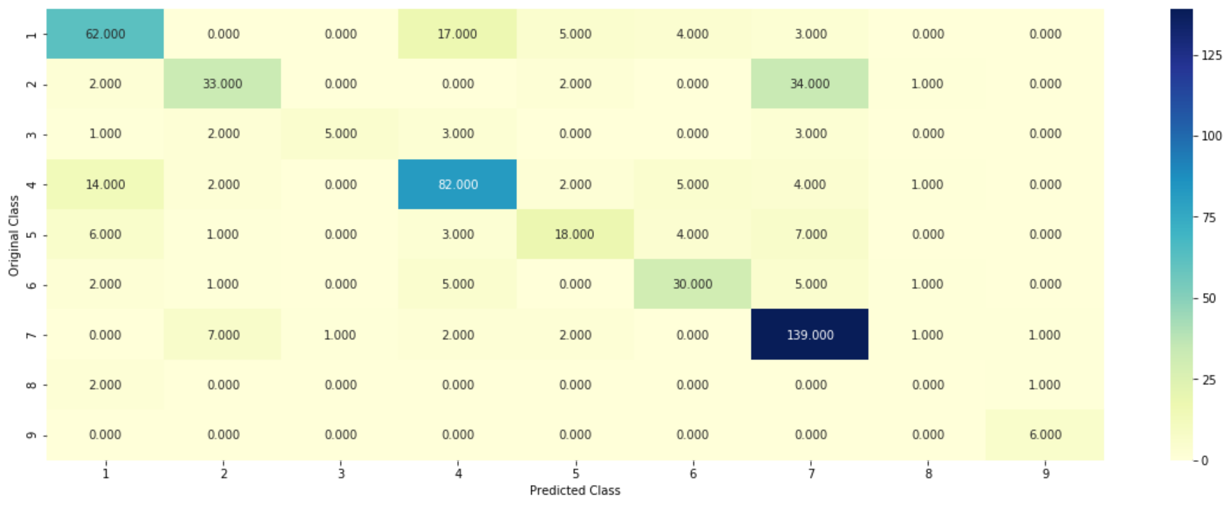


```
For values of best alpha =  0.001 The train log loss is: 0.6483218400933497
For values of best alpha =  0.001 The cross validation log loss is: 0.954147870
2481345
For values of best alpha =  0.001 The test log loss is: 0.9771034733314872
```

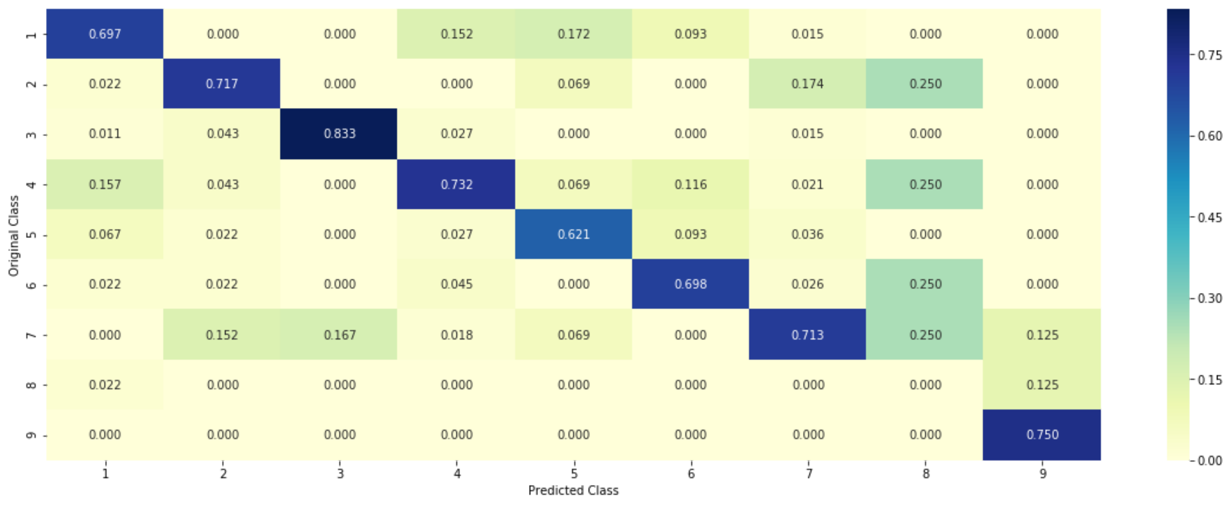
4.3.1.2. Testing the model with best hyper paramters

```
In [0]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
        # -------------------------------
        # default parameters
        # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inter
        # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
        # 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 Grad
        # predict(X)    Predict class labels for samples in X.

        #-------------------------------
        # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/less
        #------------------------------
        clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2
        predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodin
```

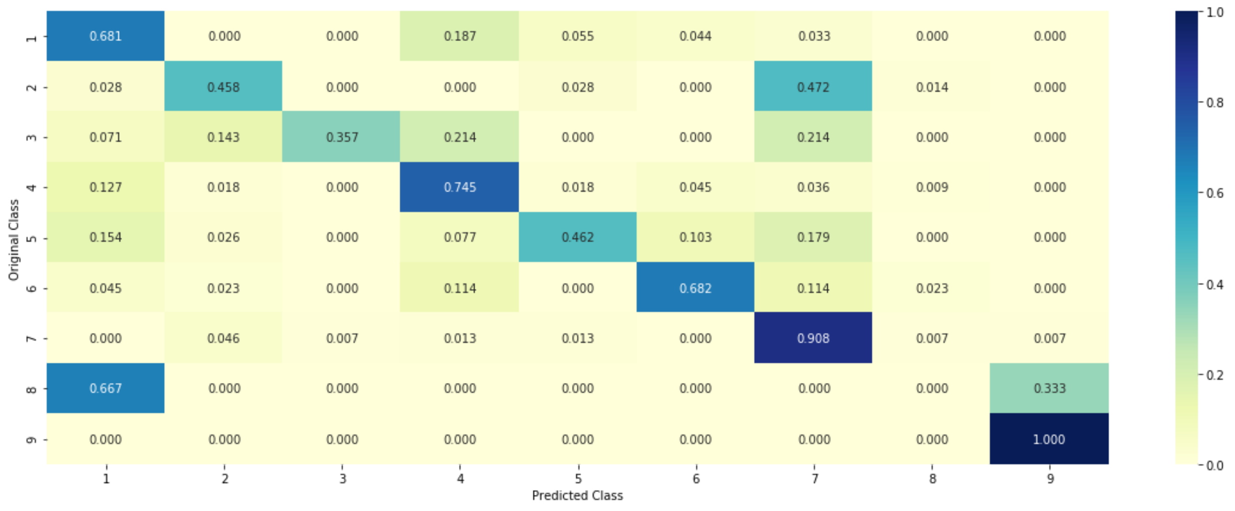
```
Log loss : 0.9541478702481345
Number of mis-classified points : 0.2951127819548872
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [0]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_n
        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 No
```

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
        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_one
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
        print("-"*50)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0063 0.5497 0.0033 0.0078 0.0302 0.0092 0.38
48 0.0081 0.0006]]
Actual Class : 2
--------------------------------------------------
```

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_one
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
        print("-"*50)
```

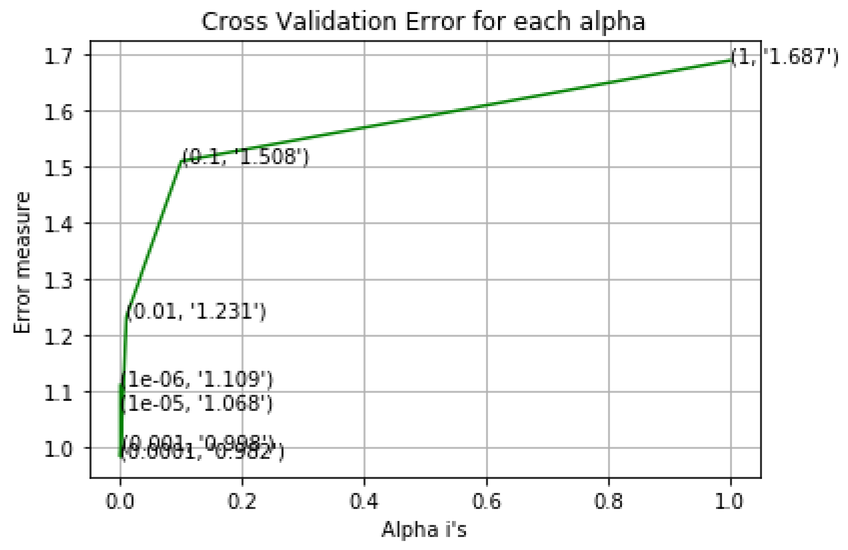
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0404 0.0621 0.0108 0.0391 0.0281 0.0169 0.79
39 0.0048 0.004 ]]
Actual Class : 7
--------------------------------------------------
```

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/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inter
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
# 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 Grad
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/less
#------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mo
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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_, 
    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_sta
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:",
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
for alpha = 1e-06
Log Loss : 1.109469275804417
for alpha = 1e-05
Log Loss : 1.0683437405676588
for alpha = 0.0001
Log Loss : 0.981756287843249
for alpha = 0.001
Log Loss : 0.9976035090112372
for alpha = 0.01
Log Loss : 1.231375738984243
for alpha = 0.1
Log Loss : 1.5078919931015102
for alpha = 1
Log Loss : 1.6872906162760626
```

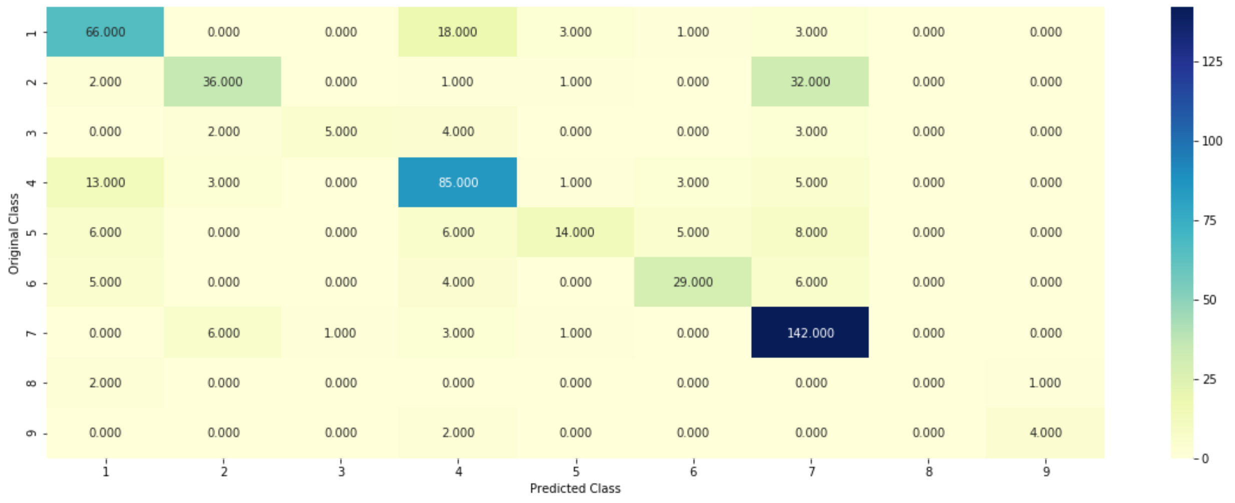


```
For values of best alpha =  0.0001 The train log loss is: 0.456388293599789
For values of best alpha =  0.0001 The cross validation log loss is: 0.98175628
7843249
For values of best alpha =  0.0001 The test log loss is: 0.9866424801154886
```

4.3.2.2. Testing model with best hyper parameters

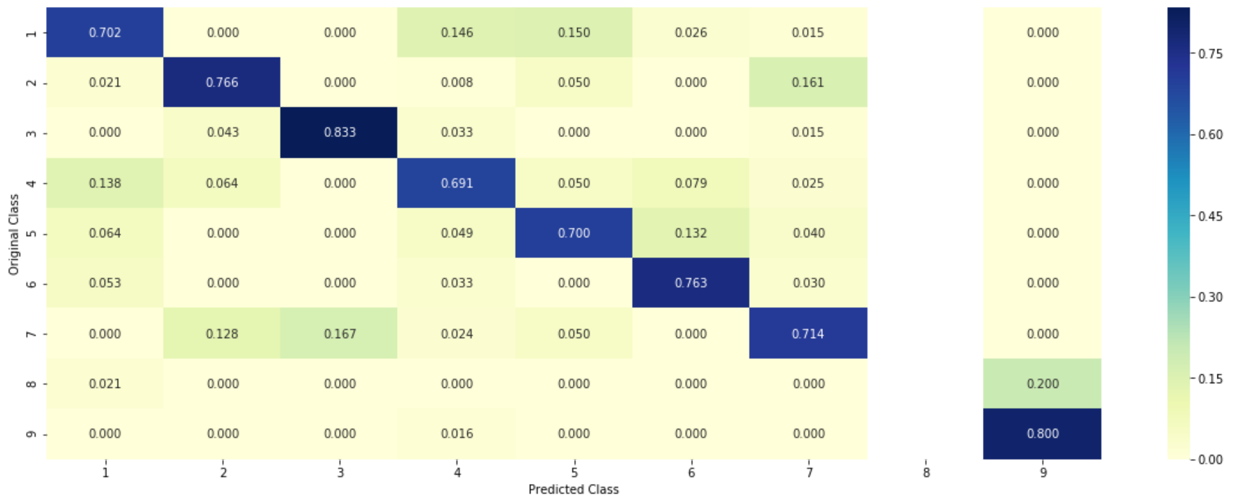
In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_sta
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodin
```

```
Log loss : 0.981756287843249
Number of mis-classified points : 0.28383458646616544
-------------------- Confusion matrix --------------------
```



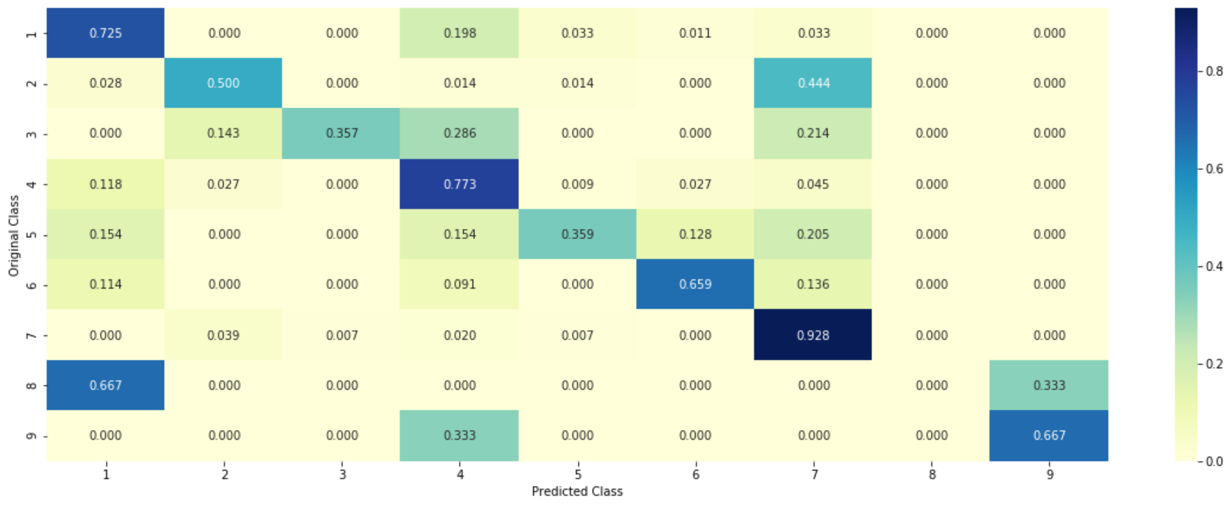
	1	2	3	4	5	6	7	8	9
1	66.000	0.000	0.000	18.000	3.000	1.000	3.000	0.000	0.000
2	2.000	36.000	0.000	1.000	1.000	0.000	32.000	0.000	0.000
3	0.000	2.000	5.000	4.000	0.000	0.000	3.000	0.000	0.000
4	13.000	3.000	0.000	85.000	1.000	3.000	5.000	0.000	0.000
5	6.000	0.000	0.000	6.000	14.000	5.000	8.000	0.000	0.000
6	5.000	0.000	0.000	4.000	0.000	29.000	6.000	0.000	0.000
7	0.000	6.000	1.000	3.000	1.000	0.000	142.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	0.000	0.000	0.000	2.000	0.000	0.000	0.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.702	0.000	0.000	0.146	0.150	0.026	0.015		0.000
2	0.021	0.766	0.000	0.008	0.050	0.000	0.161		0.000
3	0.000	0.043	0.833	0.033	0.000	0.000	0.015		0.000
4	0.138	0.064	0.000	0.691	0.050	0.079	0.025		0.000
5	0.064	0.000	0.000	0.049	0.700	0.132	0.040		0.000
6	0.053	0.000	0.000	0.033	0.000	0.763	0.030		0.000
7	0.000	0.128	0.167	0.024	0.050	0.000	0.714		0.000
8	0.021	0.000	0.000	0.000	0.000	0.000	0.000		0.200
9	0.000	0.000	0.000	0.016	0.000	0.000	0.000		0.800

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



4.3.2.3. Feature Importance, Correctly Classified point

In [0]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_sta
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_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[1.270e-02 7.029e-01 2.800e-03 8.600e-03 2.940e
-02 6.400e-03 2.286e-01
  8.300e-03 3.000e-04]]
Actual Class : 2
--------------------------------------------------
```

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_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0371 0.0424 0.012  0.0197 0.0343 0.0165 0.82
93 0.0047 0.0039]]
Actual Class : 7
--------------------------------------------------
```

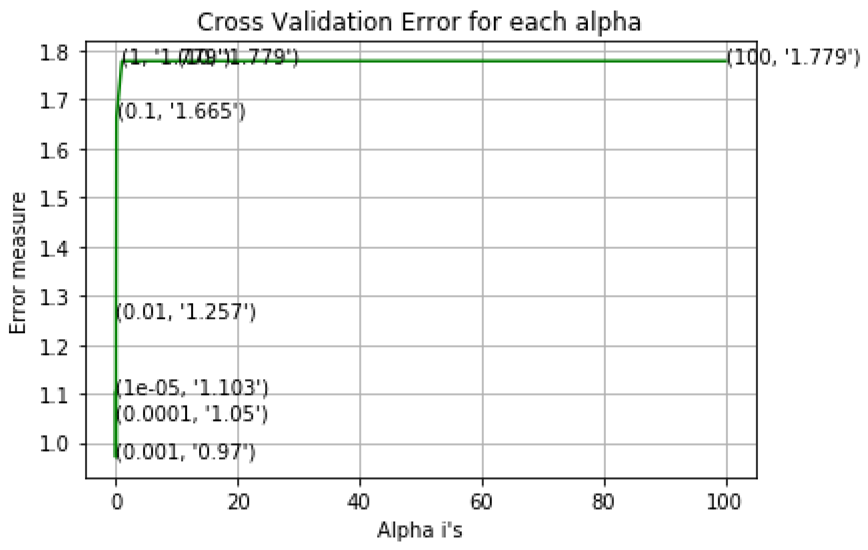
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [0]:
```
# read more about support vector machines with linear kernals here http://scikit-

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, prol
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_functio

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mo
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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='hi
    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_,
    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
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:",
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
for C = 1e-05
Log Loss : 1.1030285478134367
for C = 0.0001
Log Loss : 1.0501924182104416
for C = 0.001
Log Loss : 0.9702112724497592
for C = 0.01
Log Loss : 1.2574780041704086
for C = 0.1
Log Loss : 1.664590275567539
for C = 1
Log Loss : 1.7785275815700619
for C = 10
Log Loss : 1.7785275064282509
for C = 100
Log Loss : 1.7785275973570276
```

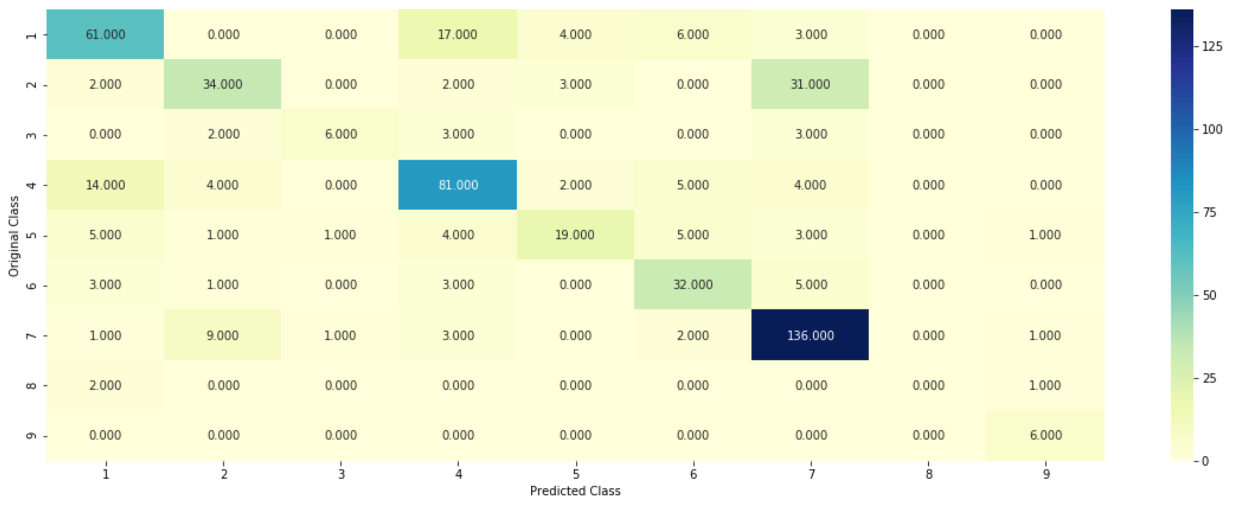


```
For values of best alpha =  0.001 The train log loss is: 0.5562551362875902
For values of best alpha =  0.001 The cross validation log loss is: 0.970211272
4497592
For values of best alpha =  0.001 The test log loss is: 1.0399317714748202
```

4.4.2. Testing model with best hyper parameters

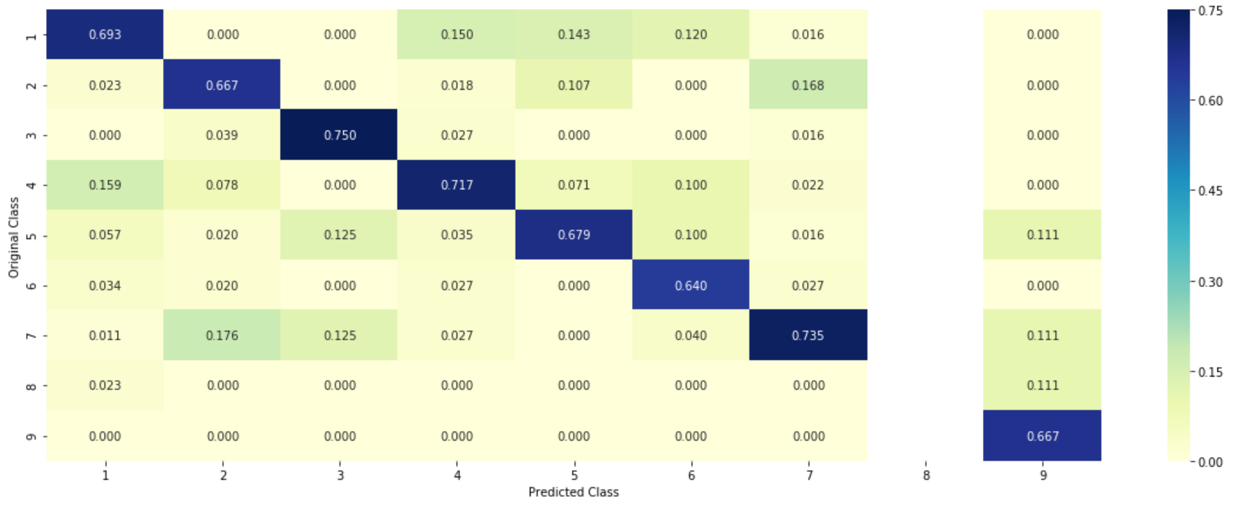
```
In [0]: # read more about support vector machines with linear kernals here http://scikit-

        # -------------------------------
        # default parameters
        # SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, pro
        # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_functio

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


        # clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='b
        clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_s
        predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding
```

```
Log loss : 0.9702112724497592
Number of mis-classified points : 0.2951127819548872
-------------------- Confusion matrix --------------------
```



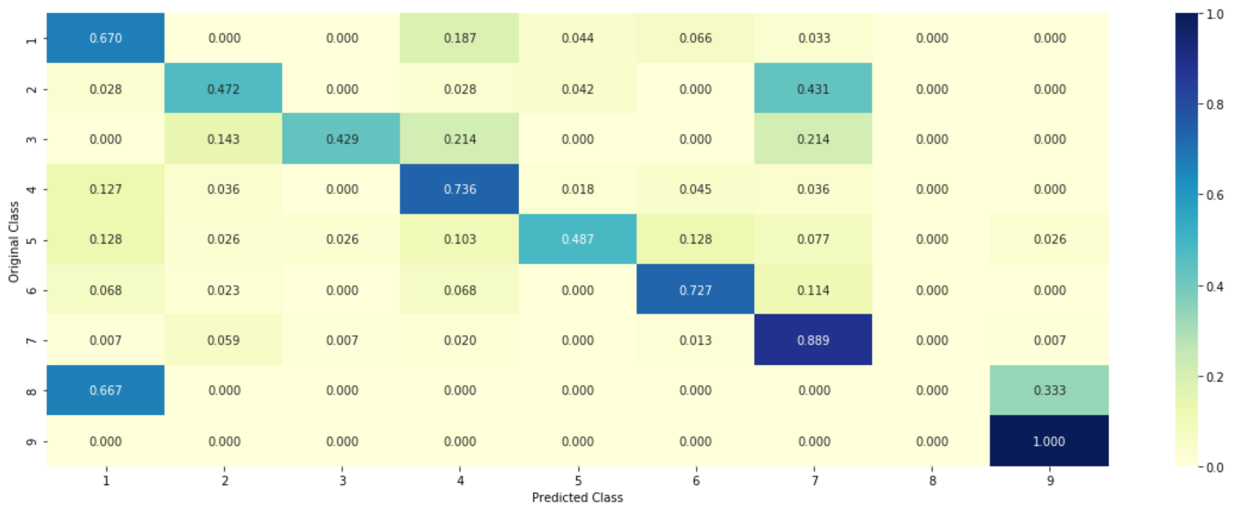
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	61.000	0.000	0.000	17.000	4.000	6.000	3.000	0.000	0.000
2	2.000	34.000	0.000	2.000	3.000	0.000	31.000	0.000	0.000
3	0.000	2.000	6.000	3.000	0.000	0.000	3.000	0.000	0.000
4	14.000	4.000	0.000	81.000	2.000	5.000	4.000	0.000	0.000
5	5.000	1.000	1.000	4.000	19.000	5.000	3.000	0.000	1.000
6	3.000	1.000	0.000	3.000	0.000	32.000	5.000	0.000	0.000
7	1.000	9.000	1.000	3.000	0.000	2.000	136.000	0.000	1.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.693	0.000	0.000	0.150	0.143	0.120	0.016		0.000
2	0.023	0.667	0.000	0.018	0.107	0.000	0.168		0.000
3	0.000	0.039	0.750	0.027	0.000	0.000	0.016		0.000
4	0.159	0.078	0.000	0.717	0.071	0.100	0.022		0.000
5	0.057	0.020	0.125	0.035	0.679	0.100	0.016		0.111
6	0.034	0.020	0.000	0.027	0.000	0.640	0.027		0.000
7	0.011	0.176	0.125	0.027	0.000	0.040	0.735		0.111
8	0.023	0.000	0.000	0.000	0.000	0.000	0.000		0.111
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000		0.667

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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [0]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_s
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_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0477 0.7184 0.0098 0.0286 0.0356 0.0136 0.13
54 0.0079 0.003 ]]
Actual Class : 2
--------------------------------------------------
```

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_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.066  0.037  0.0123 0.0246 0.0371 0.0196 0.79
52 0.0047 0.0035]]
Actual Class : 7
--------------------------------------------------
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

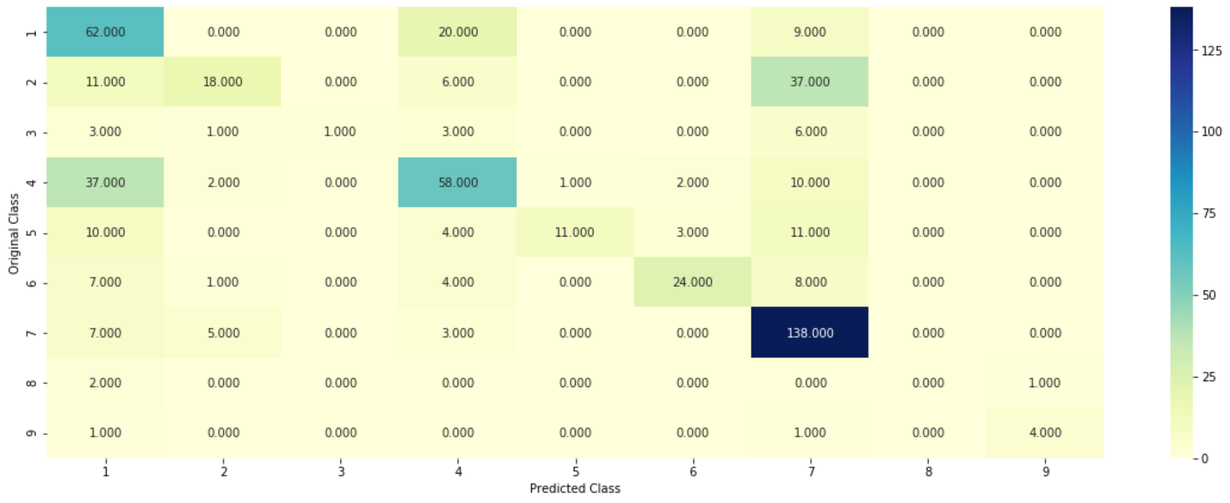
In [0]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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=
        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.classe
        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_l
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='gi
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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1957565414748372
for n_estimators = 100 and max depth =  10
Log Loss : 1.1922923251796937
for n_estimators = 200 and max depth =  5
Log Loss : 1.1833444907195354
for n_estimators = 200 and max depth =  10
Log Loss : 1.1848133910918428
for n_estimators = 500 and max depth =  5
Log Loss : 1.1751897894382282
for n_estimators = 500 and max depth =  10
Log Loss : 1.1775093298275472
for n_estimators = 1000 and max depth =  5
Log Loss : 1.175324285771641
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1759318447074925
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1746573512695804
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1750976400837914
For values of best estimator =  2000 The train log loss is: 0.9133225437684408
For values of best estimator =  2000 The cross validation log loss is: 1.174657
3512695804
For values of best estimator =  2000 The test log loss is: 1.1843228074266527
```

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

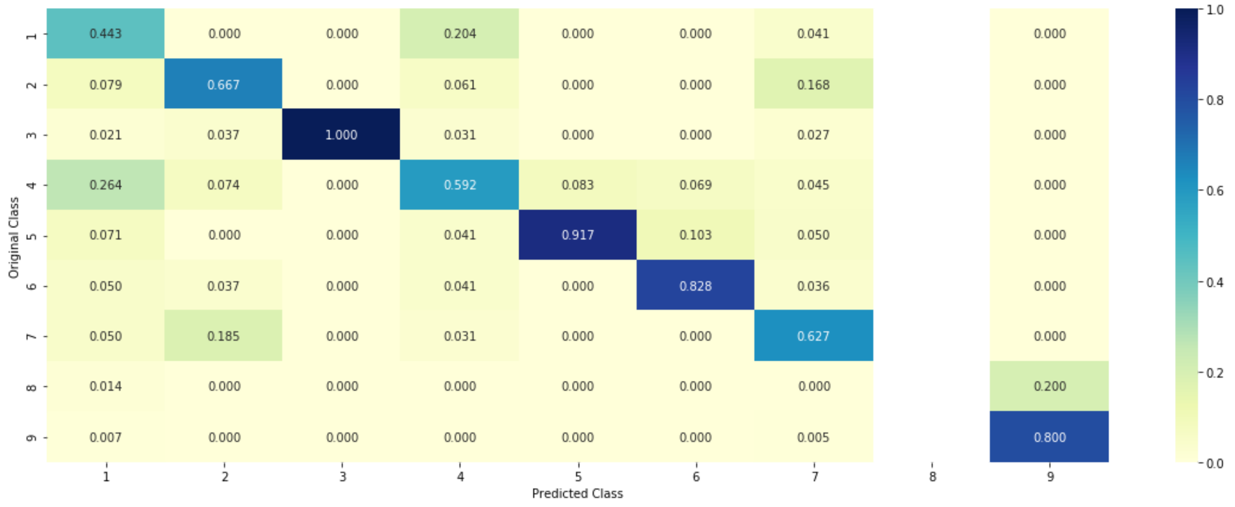
In [0]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gi
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding
```

```
Log loss : 1.1746573512695804
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```

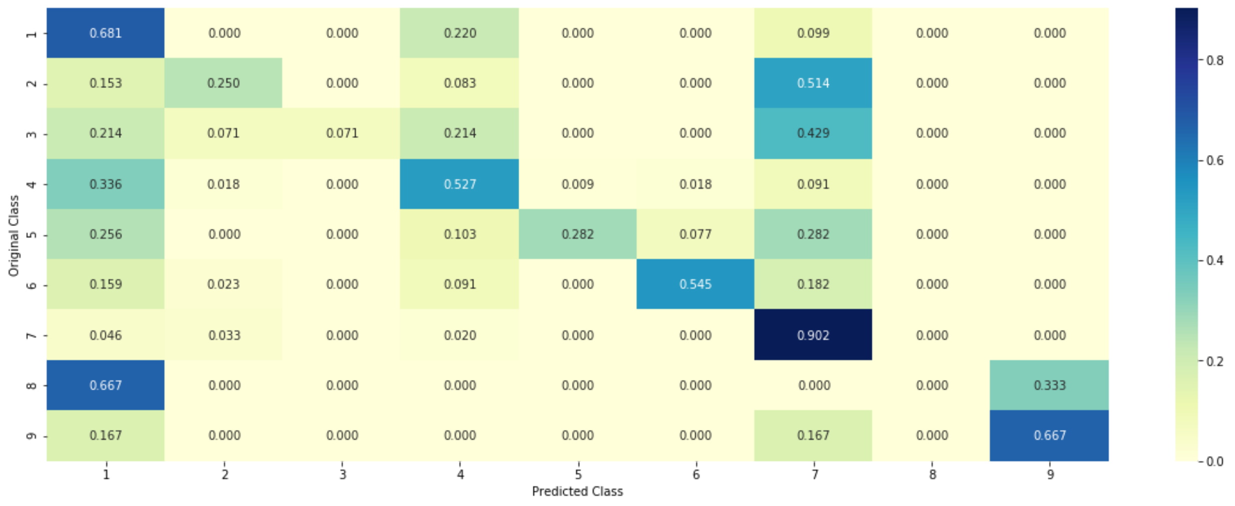


	1	2	3	4	5	6	7	8	9
1	62.000	0.000	0.000	20.000	0.000	0.000	9.000	0.000	0.000
2	11.000	18.000	0.000	6.000	0.000	0.000	37.000	0.000	0.000
3	3.000	1.000	1.000	3.000	0.000	0.000	6.000	0.000	0.000
4	37.000	2.000	0.000	58.000	1.000	2.000	10.000	0.000	0.000
5	10.000	0.000	0.000	4.000	11.000	3.000	11.000	0.000	0.000
6	7.000	1.000	0.000	4.000	0.000	24.000	8.000	0.000	0.000
7	7.000	5.000	0.000	3.000	0.000	0.000	138.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	4.000

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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [0]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gi
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_one
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0321 0.3367 0.0178 0.0319 0.0469 0.037  0.48
84 0.0068 0.0024]]
Actual Class : 2
--------------------------------------------------
```

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_one
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0506 0.174  0.0193 0.0494 0.0519 0.0435 0.60
19 0.0066 0.0029]]
Actuall Class : 7
--------------------------------------------------
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [0]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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=
        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.classe
        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_l
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='gi
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 los
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validat
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss
```

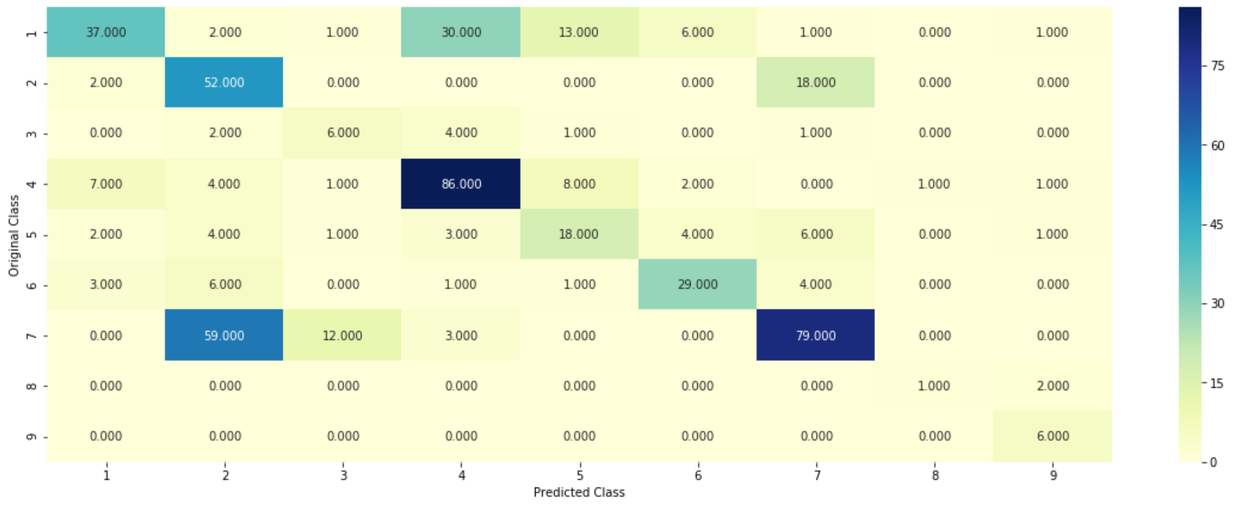
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.252395114049456
for n_estimators = 10 and max depth =  3
Log Loss : 1.731497744533845
for n_estimators = 10 and max depth =  5
Log Loss : 1.3226414368397983
for n_estimators = 10 and max depth =  10
Log Loss : 2.0136420928338534
for n_estimators = 50 and max depth =  2
Log Loss : 1.752653605603669
for n_estimators = 50 and max depth =  3
Log Loss : 1.4520872336624988
for n_estimators = 50 and max depth =  5
Log Loss : 1.2873101045833217
for n_estimators = 50 and max depth =  10
Log Loss : 1.7841439204453657
for n_estimators = 100 and max depth =  2
Log Loss : 1.5728644081443437
for n_estimators = 100 and max depth =  3
Log Loss : 1.5258605820806055
for n_estimators = 100 and max depth =  5
Log Loss : 1.2615756094201513
for n_estimators = 100 and max depth =  10
Log Loss : 1.7541425342240604
for n_estimators = 200 and max depth =  2
Log Loss : 1.6622199505325508
for n_estimators = 200 and max depth =  3
Log Loss : 1.5578686388512741
for n_estimators = 200 and max depth =  5
Log Loss : 1.3670922590377885
for n_estimators = 200 and max depth =  10
Log Loss : 1.7883157999609138
for n_estimators = 500 and max depth =  2
Log Loss : 1.7500832413184833
for n_estimators = 500 and max depth =  3
Log Loss : 1.6102893559095166
for n_estimators = 500 and max depth =  5
Log Loss : 1.3780588483750758
```

```
for n_estimators = 500 and max depth =  10
Log Loss : 1.8537106360768496
for n_estimators = 1000 and max depth =  2
Log Loss : 1.711888119815348
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6059204768943431
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3557250699593748
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8467500422905054
For values of best alpha =  100 The train log loss is: 0.062364610913203444
For values of best alpha =  100 The cross validation log loss is: 1.26157560942
01513
For values of best alpha =  100 The test log loss is: 1.2463169929506979
```

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

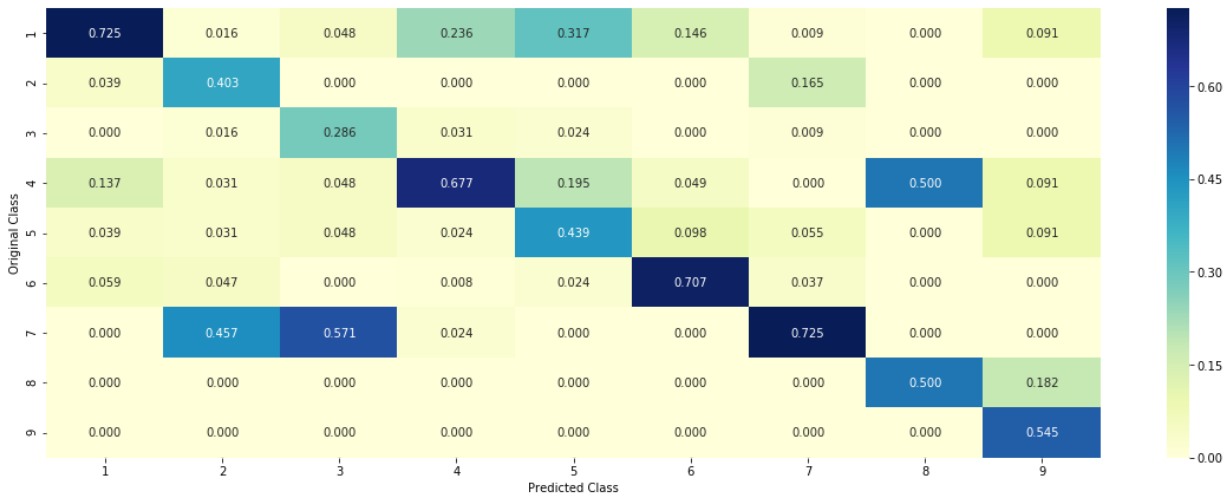
In [0]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCo
```

```
Log loss : 1.2615756094201513
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```



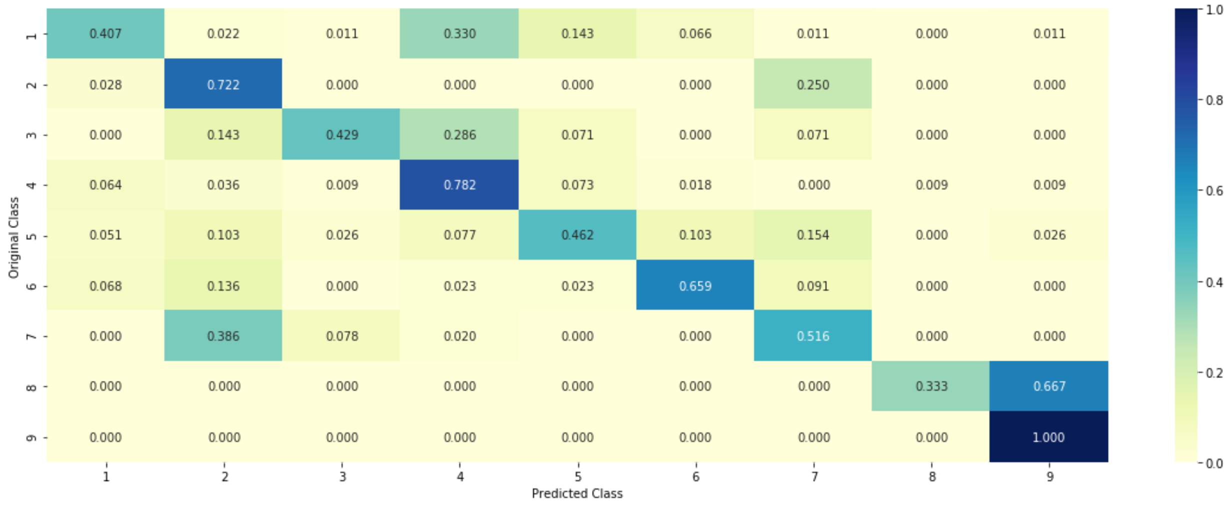
Original \ Predicted	1	2	3	4	5	6	7	8	9
1	37.000	2.000	1.000	30.000	13.000	6.000	1.000	0.000	1.000
2	2.000	52.000	0.000	0.000	0.000	0.000	18.000	0.000	0.000
3	0.000	2.000	6.000	4.000	1.000	0.000	1.000	0.000	0.000
4	7.000	4.000	1.000	86.000	8.000	2.000	0.000	1.000	1.000
5	2.000	4.000	1.000	3.000	18.000	4.000	6.000	0.000	1.000
6	3.000	6.000	0.000	1.000	1.000	29.000	4.000	0.000	0.000
7	0.000	59.000	12.000	3.000	0.000	0.000	79.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.725	0.016	0.048	0.236	0.317	0.146	0.009	0.000	0.091
2	0.039	0.403	0.000	0.000	0.000	0.000	0.165	0.000	0.000
3	0.000	0.016	0.286	0.031	0.024	0.000	0.009	0.000	0.000
4	0.137	0.031	0.048	0.677	0.195	0.049	0.000	0.500	0.091
5	0.039	0.031	0.048	0.024	0.439	0.098	0.055	0.000	0.091
6	0.059	0.047	0.000	0.008	0.024	0.707	0.037	0.000	0.000
7	0.000	0.457	0.571	0.024	0.000	0.000	0.725	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.182
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.545

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



	1	2	3	4	5	6	7	8	9
1	0.407	0.022	0.011	0.330	0.143	0.066	0.011	0.000	0.011
2	0.028	0.722	0.000	0.000	0.000	0.000	0.250	0.000	0.000
3	0.000	0.143	0.429	0.286	0.071	0.000	0.071	0.000	0.000
4	0.064	0.036	0.009	0.782	0.073	0.018	0.000	0.009	0.009
5	0.051	0.103	0.026	0.077	0.462	0.103	0.154	0.000	0.026
6	0.068	0.136	0.000	0.023	0.023	0.659	0.091	0.000	0.000
7	0.000	0.386	0.078	0.020	0.000	0.000	0.516	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [0]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gi
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
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_res
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0106 0.7461 0.0425 0.0136 0.0127 0.0277 0.11
46 0.0242 0.0081]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation 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
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

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
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_res
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: [[0.0237 0.195  0.2041 0.0223 0.0324 0.0451 0.40
41 0.0489 0.0243]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation 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
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

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/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
# 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 Grad
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/less
#------------------------------



# read more about support vector machines with linear kernals here http://scikit-
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, prob
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function

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



# read more about support vector machines with linear kernals here http://scikit-
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_d
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/less
# --------------------------------


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

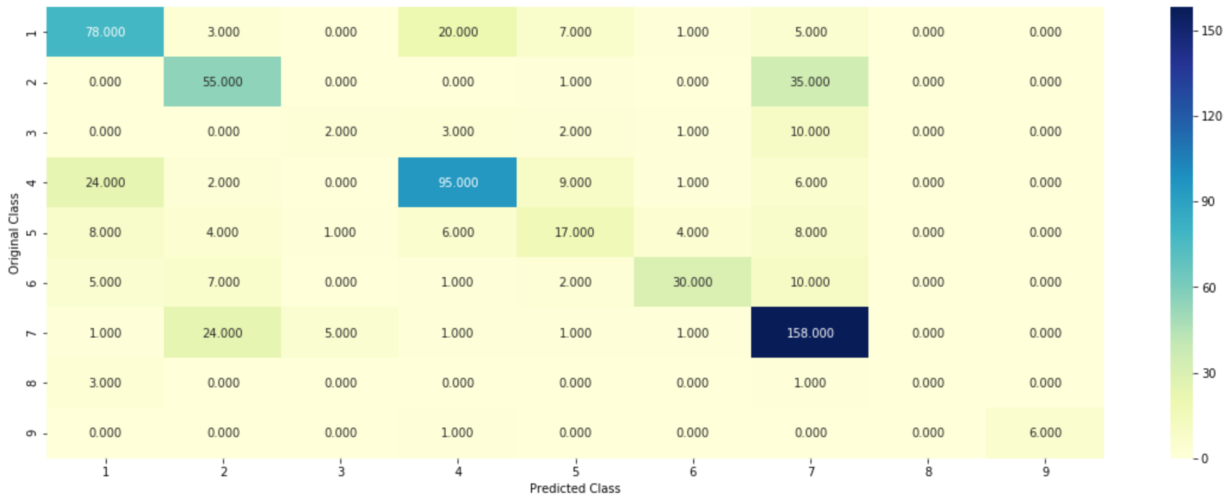
```
clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced'
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.pred
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv
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_cl
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i,
    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: 0.96
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.17
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.485
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.063
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.180
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.521
```

4.7.2 testing the model with the best hyper parameters

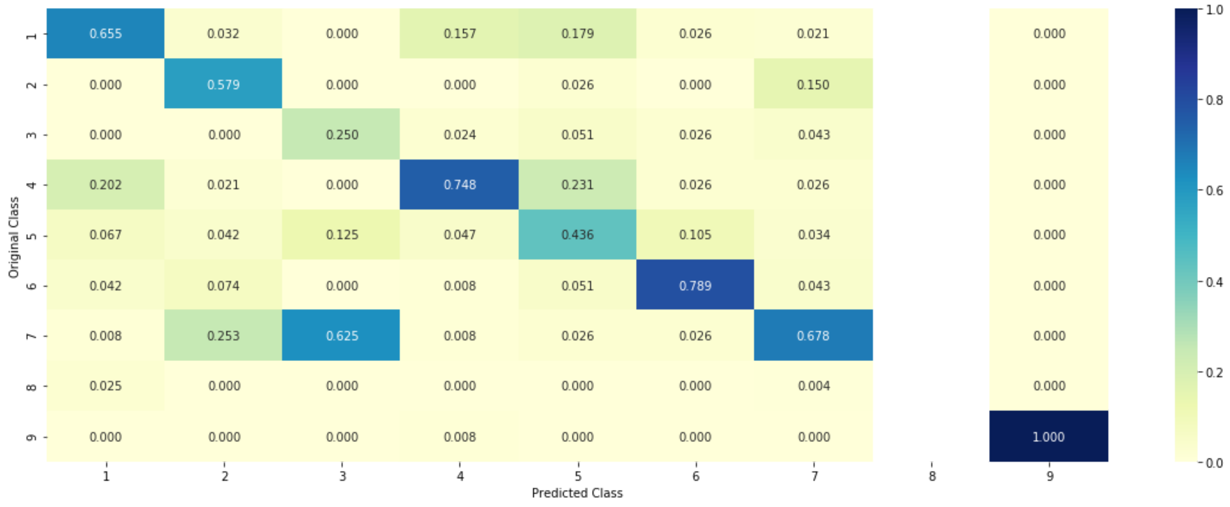
```
In [0]: lr = LogisticRegression(C=0.1)
        sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classi
        sclf.fit(train_x_onehotCoding, train_y)

        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_o
        plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.6105009232885991
Log loss (CV) on the stacking classifier : 1.0626084032071585
Log loss (test) on the stacking classifier : 1.0836944361840508
Number of missclassified point : 0.3368421052631579
-------------------- Confusion matrix --------------------
```



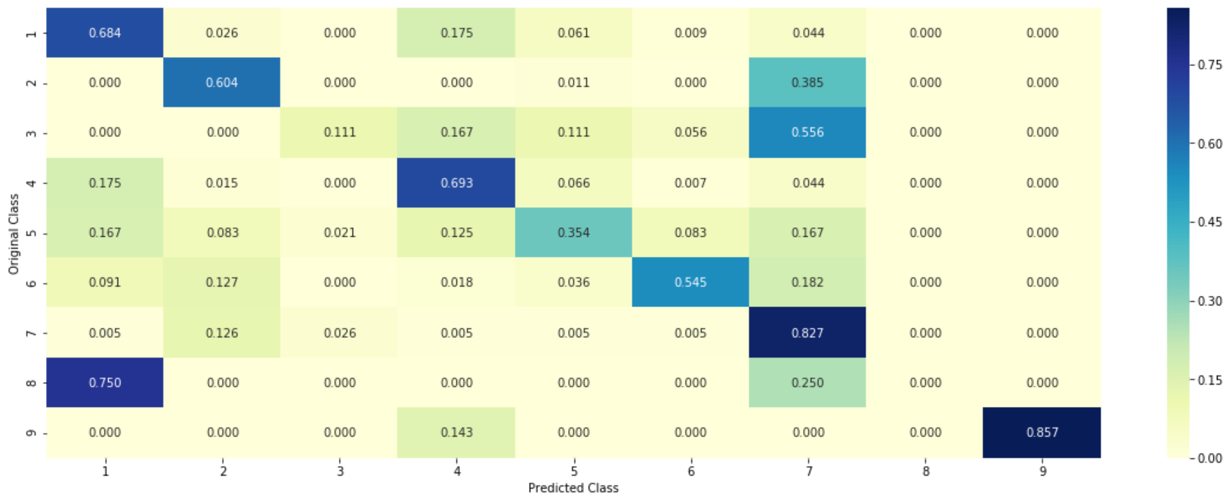
	1	2	3	4	5	6	7	8	9
1	78.000	3.000	0.000	20.000	7.000	1.000	5.000	0.000	0.000
2	0.000	55.000	0.000	0.000	1.000	0.000	35.000	0.000	0.000
3	0.000	0.000	2.000	3.000	2.000	1.000	10.000	0.000	0.000
4	24.000	2.000	0.000	95.000	9.000	1.000	6.000	0.000	0.000
5	8.000	4.000	1.000	6.000	17.000	4.000	8.000	0.000	0.000
6	5.000	7.000	0.000	1.000	2.000	30.000	10.000	0.000	0.000
7	1.000	24.000	5.000	1.000	1.000	1.000	158.000	0.000	0.000
8	3.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.655	0.032	0.000	0.157	0.179	0.026	0.021		0.000
2	0.000	0.579	0.000	0.000	0.026	0.000	0.150		0.000
3	0.000	0.000	0.250	0.024	0.051	0.026	0.043		0.000
4	0.202	0.021	0.000	0.748	0.231	0.026	0.026		0.000
5	0.067	0.042	0.125	0.047	0.436	0.105	0.034		0.000
6	0.042	0.074	0.000	0.008	0.051	0.789	0.043		0.000
7	0.008	0.253	0.625	0.008	0.026	0.026	0.678		0.000
8	0.025	0.000	0.000	0.000	0.000	0.000	0.004		0.000
9	0.000	0.000	0.000	0.008	0.000	0.000	0.000		1.000

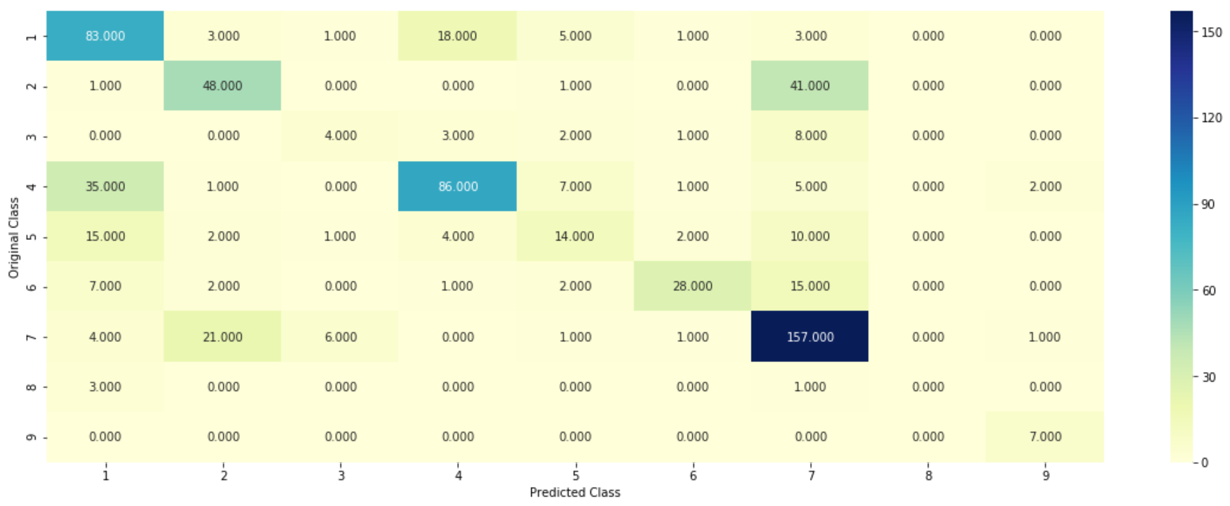
```
-------------------- Recall matrix (Row sum=1) --------------------
```



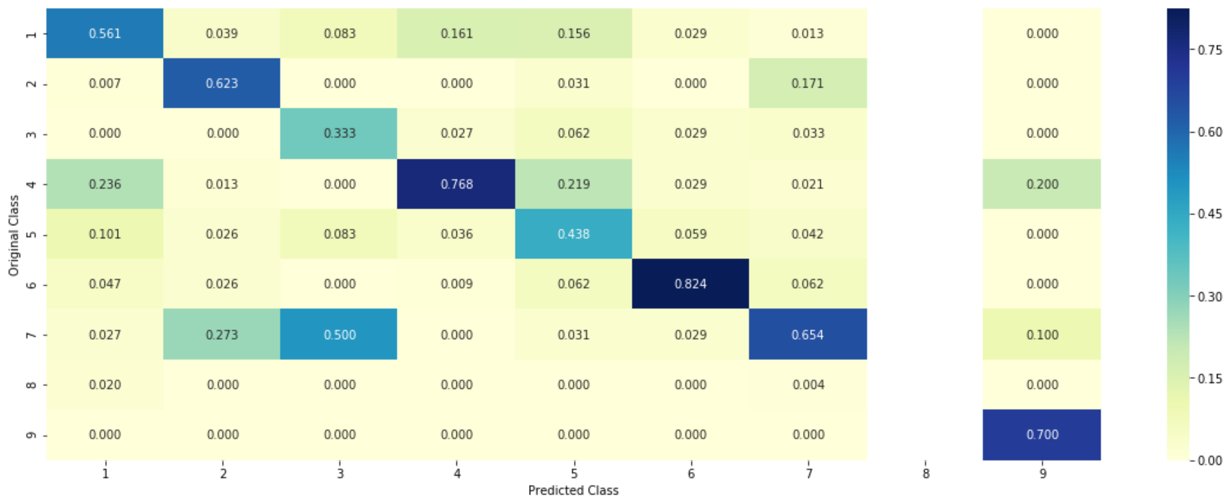
4.7.3 Maximum Voting classifier

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

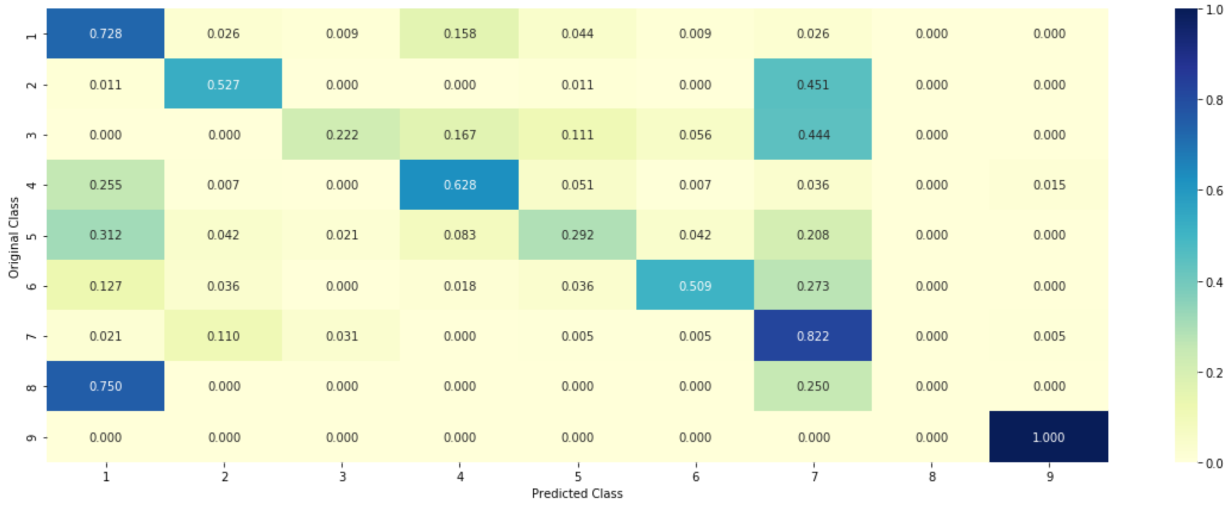
```
Log loss (train) on the VotingClassifier : 0.8620517996179623
Log loss (CV) on the VotingClassifier : 1.1384625629157292
Log loss (test) on the VotingClassifier : 1.1335406160107757
Number of missclassified point : 0.35789473684210527
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.728	0.026	0.009	0.158	0.044	0.009	0.026	0.000	0.000
2	0.011	0.527	0.000	0.000	0.011	0.000	0.451	0.000	0.000
3	0.000	0.000	0.222	0.167	0.111	0.056	0.444	0.000	0.000
4	0.255	0.007	0.000	0.628	0.051	0.007	0.036	0.000	0.015
5	0.312	0.042	0.021	0.083	0.292	0.042	0.208	0.000	0.000
6	0.127	0.036	0.000	0.018	0.036	0.509	0.273	0.000	0.000
7	0.021	0.110	0.031	0.000	0.005	0.005	0.822	0.000	0.005
8	0.750	0.000	0.000	0.000	0.000	0.000	0.250	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

In [2]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No","Model","Train_logloss","Cv_logloss","Test_logloss", "Mis

x.add_row(["1","Naive Bayes","0.760","1.158","1.188","0.36"])
x.add_row(["2","KNN","0.718","0.978","1.055","0.33"])
x.add_row(["3","Logistic regression with class balancing","0.648","0.95","0.977",
x.add_row(["4","Logistic regression without class balancing","0.456","0.981","0.9
x.add_row(["5","Linear svm(with one hot encoding)","0.55","0.97","1.03","0.30"])
x.add_row(["6","Random Forest(with one hot encoding)","0.97","1.17","1.18","0.40"
x.add_row(["7","Random Forest(with response coding)","0.062","1.26","1.24","0.41"
x.add_row(["8","Stacking classifier","0.61","1.06","1.08","0.33"])
x.add_row(["9","Maximum voting classifier","0.86","1.138","1.33","0.35"])


print(x)
```

```
+------+---------------------------------------------+---------------+---------
---+--------------+---------------------+
| S.No |                    Model                    | Train_logloss | Cv_loglo
ss | Test_logloss | Misclassified_error |
+------+---------------------------------------------+---------------+---------
---+--------------+---------------------+
|  1   |                 Naive Bayes                 |     0.760     |   1.158
|    1.188     |         0.36        |
|  2   |                     KNN                     |     0.718     |   0.978
|    1.055     |         0.33        |
|  3   |   Logistic regression with class balancing  |     0.648     |    0.95
|    0.977     |         0.30        |
|  4   | Logistic regression without class balancing |     0.456     |   0.981
|    0.986     |         0.28        |
|  5   |       Linear svm(with one hot encoding)     |      0.55     |    0.97
|     1.03     |         0.30        |
|  6   |     Random Forest(with one hot encoding)    |      0.97     |    1.17
|     1.18     |         0.40        |
|  7   |     Random Forest(with response coding)     |     0.062     |    1.26
|     1.24     |         0.41        |
|  8   |             Stacking classifier             |      0.61     |    1.06
|     1.08     |         0.33        |
|  9   |          Maximum voting classifier          |      0.86     |   1.138
|     1.33     |         0.35        |
+------+---------------------------------------------+---------------+---------
---+--------------+---------------------+
```

Conclusion:

Feature enginnering steps:

1. At first we are taking the Gene & Variation features and combine both in Gene_Var_data corpus.

2. Then build TF_IDF veactorizer on top of Gene_Var_data fit that by using TFIDF transform into Train, Test, cv.
3. By doing the above feature engineering steps we can have some more information to train our model.
4. After Feature engineering misclassified errors are quite low when compared to without feature engineered models.