Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25

2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y

ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
!pip install nltk
!pip install mlxtend
!pip install seaborn
!pip install xgboost
#!pip install imblearn
```

```
Collecting nltk
  Downloading https://files.pythonhosted.org/packages/50/09/3b1755d528a
d9156ee7243d52aa5cd2b809ef053a0f31b53d92853dd653a/nltk-3.3.0.zip (1.4M
B)
    100% |################################| 1.4MB 7.1MB/s ta 0:00:01
Requirement already satisfied: six in /opt/conda/envs/py3.6/lib/python
3.6/site-packages (from nltk) (1.11.0)
Building wheels for collected packages: nltk
  Running setup.py bdist_wheel for nltk ... done
  Stored in directory: /home/jovyan/.cache/pip/wheels/d1/ab/40/3bceea46
922767e42986aef7606a600538ca80de6062dc266c
Successfully built nltk
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
```

```
 have widgetsnbextension 3.2.1 which is incompatible.
Installing collected packages: nltk
Successfully installed nltk-3.3
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
Collecting mlxtend
  Downloading https://files.pythonhosted.org/packages/d0/f9/798cb32550d
cbc9e0e3c143dc7144d2631df171423ed143cdb8b38ee2e5e/mlxtend-0.13.0-py2.py
3-none-any.whl (1.3MB)
    100% |################################| 1.3MB 8.0MB/s ta 0:00:011
Requirement already satisfied: matplotlib>=1.5.1 in /opt/conda/envs/py
3.6/lib/python3.6/site-packages (from mlxtend) (2.1.2)
Requirement already satisfied: pandas>=0.17.1 in /opt/conda/envs/py3.6/
lib/python3.6/site-packages (from mlxtend) (0.20.3)
Requirement already satisfied: numpy>=1.10.4 in /opt/conda/envs/py3.6/l
ib/python3.6/site-packages (from mlxtend) (1.12.1)
Requirement already satisfied: setuptools in /opt/conda/envs/py3.6/lib/
python3.6/site-packages (from mlxtend) (36.4.0)
Requirement already satisfied: scipy>=0.17 in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from mlxtend) (0.19.1)
Requirement already satisfied: scikit-learn>=0.18 in /opt/conda/envs/py
3.6/lib/python3.6/site-packages (from mlxtend) (0.19.0)
Requirement already satisfied: six>=1.10 in /opt/conda/envs/py3.6/lib/p
ython3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (1.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/envs/
py3.6/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (2.
7.3)
Requirement already satisfied: pytz in /opt/conda/envs/py3.6/lib/python
3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (2018.4)
Requirement already satisfied: cycler>=0.10 in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from matplotlib>
=1.5.1->mlxtend) (2.2.0)
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
 have widgetsnbextension 3.2.1 which is incompatible.
Installing collected packages: mlxtend
```

```
Successfully installed mlxtend-0.13.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
Collecting seaborn
  Downloading https://files.pythonhosted.org/packages/a8/76/220ba442045
9d9c4c9c9587c6ce607bf56c25b3d3d2de62056efe482dadc/seaborn-0.9.0-py3-non
e-any.whl (208kB)
    100% |################################| 215kB 7.1MB/s ta 0:00:01
Requirement already satisfied: pandas>=0.15.2 in /opt/conda/envs/py3.6/
lib/python3.6/site-packages (from seaborn) (0.20.3)
Requirement already satisfied: numpy>=1.9.3 in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from seaborn) (1.12.1)
Requirement already satisfied: scipy>=0.14.0 in /opt/conda/envs/py3.6/l
ib/python3.6/site-packages (from seaborn) (0.19.1)
Requirement already satisfied: matplotlib>=1.4.3 in /opt/conda/envs/py
3.6/lib/python3.6/site-packages (from seaborn) (2.1.2)
Requirement already satisfied: python-dateutil>=2 in /opt/conda/envs/py
3.6/lib/python3.6/site-packages (from pandas>=0.15.2->seaborn) (2.7.3)
Requirement already satisfied: pytz>=2011k in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from pandas>=0.15.2->seaborn) (2018.4)
Requirement already satisfied: six>=1.10 in /opt/conda/envs/py3.6/lib/p
ython3.6/site-packages (from matplotlib>=1.4.3->seaborn) (1.11.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from matplotlib>=1.4.3->seaborn) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from matplotlib>
=1.4.3->seaborn) (2.2.0)
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
 have widgetsnbextension 3.2.1 which is incompatible.
Installing collected packages: seaborn
Successfully installed seaborn-0.9.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
Collecting xgboost
  Downloading https://files.pythonhosted.org/packages/c6/1e/6d13dacd1d5
ea3273210162292e818f82629328ce51cdb7eb633f03e0b52/xgboost-0.80.tar.gz
```

```
 (595kB)
    100% |################################| 604kB 5.5MB/s ta 0:00:01
Requirement already satisfied: numpy in /opt/conda/envs/py3.6/lib/pytho
n3.6/site-packages (from xgboost) (1.12.1)
Requirement already satisfied: scipy in /opt/conda/envs/py3.6/lib/pytho
n3.6/site-packages (from xgboost) (0.19.1)
Building wheels for collected packages: xgboost
  Running setup.py bdist_wheel for xgboost ... done
  Stored in directory: /home/jovyan/.cache/pip/wheels/82/2d/9a/d9014a0a
dd86cdad990022418a905ca9e93948ee8862ae755d
Successfully built xgboost
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
 have widgetsnbextension 3.2.1 which is incompatible.
Installing collected packages: xgboost
Successfully installed xgboost-0.80
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
```

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
#from imblearn.over_sampling import SMOTE
from collections import Counter
```

```
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import nltk
nltk.download('stopwords')
```

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

```
/opt/conda/envs/py3.6/lib/python3.6/site-packages/sklearn/cross_validat
ion.py:41: DeprecationWarning: This module was deprecated in version 0.
18 in favor of the model_selection module into which all the refactored
classes and functions are moved. Also note that the interface of the ne
w CV iterators are different from that of this module. This module will
be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Out[2]: True

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [3]:
```
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[3]:

	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 [4]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names
=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[4]:

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

In [5]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[5]:

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

	ID	Gene	Variation	Class	TEXT
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3 Feature Engineering

Techniques obtained from these kernels/blogs.

1. https://www.kaggle.com/osciiart/redefining-treatment-0-57456-modified
2. https://www.kaggle.com/lalitparihar44/detailed-text-based-feature-engineering
3. https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/

Gene + Variation Feature

In [6]:
```
result['Gene_Variation'] = result['Gene'] + " " + result["Variation"]
result.head()
```

Out[6]:

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

	ID	Gene	Variation	Class	TEXT	Gene_Variation

Gene Count Feature

In [7]:
```
result['Gene_Share'] = result.apply(lambda r: sum([1 for w in r['Gene']
.split() if w in r['TEXT'].split()]), axis=1)
result.head()
```

Out[7]:

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

Variation Count Feature

In [8]:
```
result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['V
ariation'].split(' ') if w in r['TEXT'].split(' ')]), axis=1)
result["Variation_Share"].value_counts()
```

Out[8]:
```
1    1672
0    1577
2      58
3      10
5       2
4       2
Name: Variation_Share, dtype: int64
```

Count of Words Feature

In [9]:
```
result["Word_Count"] = result["TEXT"].apply(lambda x: len(x.split()))
result.head()
```

Out[9]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Sh
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Sh
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1

Text Count > 5000 Yes or no feature

In [10]:
```
result["Word_Count_5000"] = result["Word_Count"].apply(lambda x: 1 if x
 > 5000 else 0)
result.head()
```

Out[10]:

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

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

Character Count Feature

In [11]:
```
result['Character_Count'] = result['TEXT'].apply(lambda x: len(str(x)))
result.head()
```

Out[11]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Sh

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

Average Length of Words used in statements

In [12]:
```
result['Avg_length'] = result['Character_Count'] / result['Word_Count']
result.head()
```

Out[12]:

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

3.1.4. Preprocessing of text

In [13]:
```
# 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 t
he data
            if not word in stop_words:
                string += word + " "

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

In [14]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    nlp_preprocessing(row['TEXT'], index, 'TEXT')
print('Time took for preprocessing the text :',time.clock() - start_tim
e, "seconds")
```

```
Time took for preprocessing the text : 133.455471 seconds
```

In [15]:
```
#removing unprocessed "TEXT" from results
result.drop("TEXT", axis=1, inplace=True)
```

In [16]:
```
# Joining Text which is processed :
result = pd.merge(result, data_text,on='ID', how='left')
result.head()
```

Out[16]:

	ID	Gene	Variation	Class	Gene_Variation	Gene_Share	Variation_Share	Word_Co
0	0	FAM58A	Truncating Mutations	1	FAM58A Truncating Mutations	1	1	6089
1	1	CBL	W802*	2	CBL W802*	1	1	5722
2	2	CBL	Q249E	2	CBL Q249E	1	1	5722
3	3	CBL	N454D	3	CBL N454D	1	1	5572
4	4	CBL	L399V	4	CBL L399V	1	1	6202

3.1.5. Test, Train and Cross Validation Split

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

In [17]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
result.Gene_Variation = result.Gene_Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution o
f output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
ratify=y_true, test_size=0.2, random_state = 1)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, str
atify=y_train, test_size=0.2, random_state = 1)
```

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

In [18]:
```
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
])
```

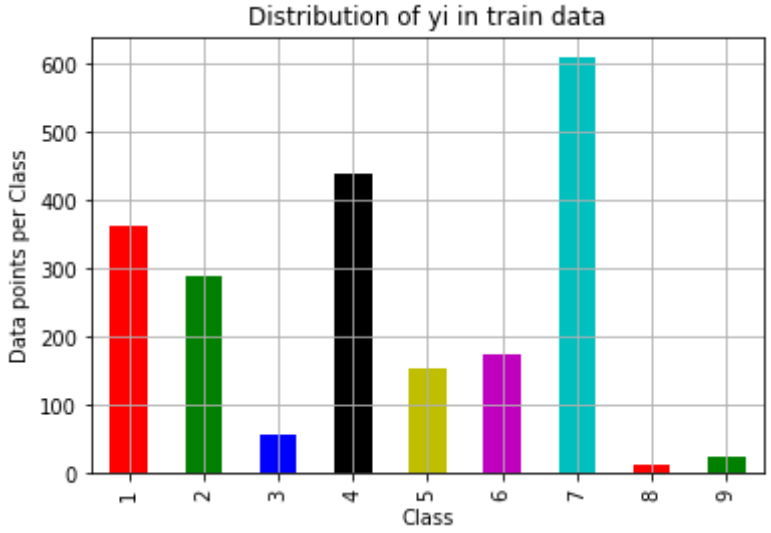
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

In [19]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
```

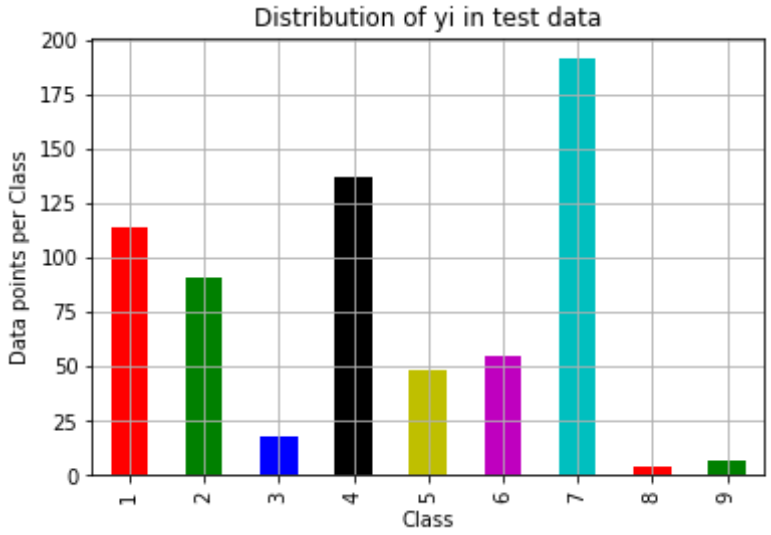
```
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distrib
ution.values[i], '(', np.round((test_class_distribution.values[i]/test_
```

```
df.shape[0]*100), 3), '%)')

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

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

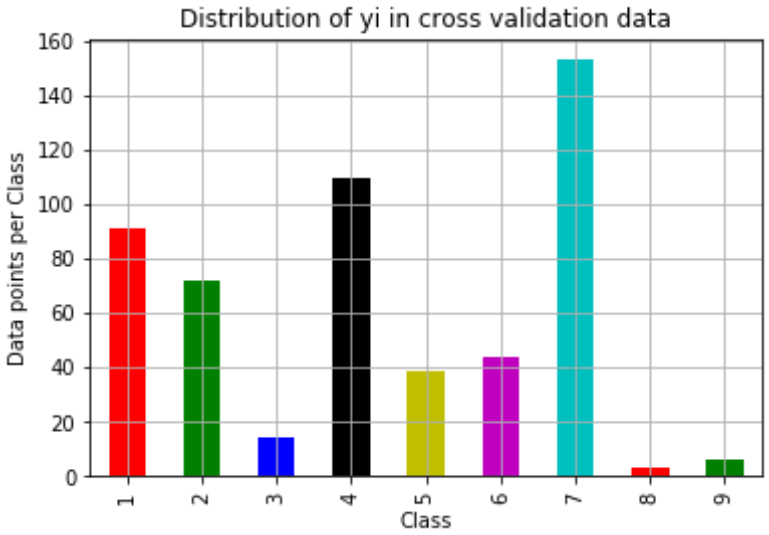



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
```

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




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




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

3.2 Prediction using a 'Random' Model

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

In [20]:
```
# 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 cl
ass i are predicted class j
```

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

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

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

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

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

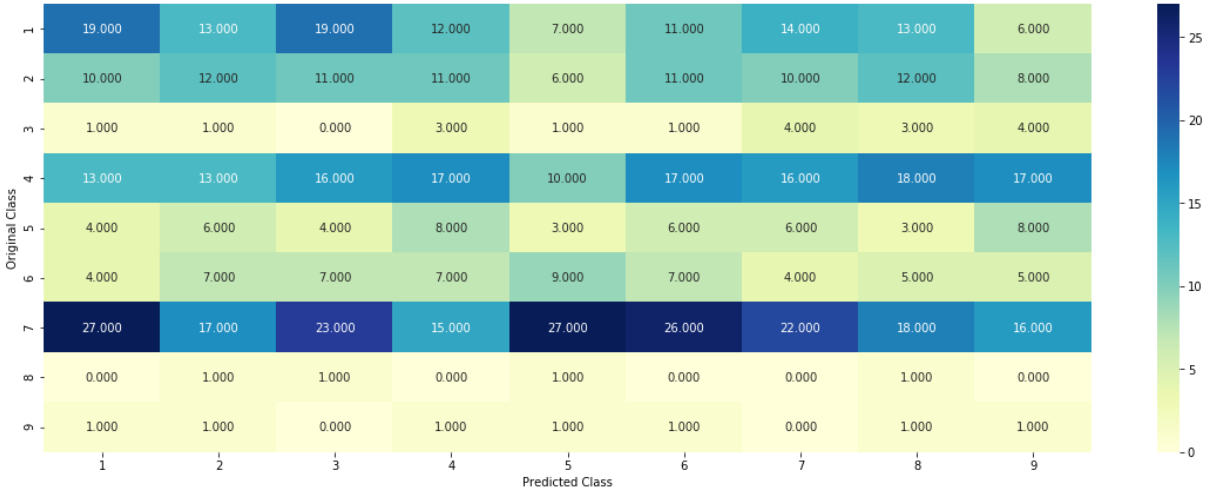
```
    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

In [21]:

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

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


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
```

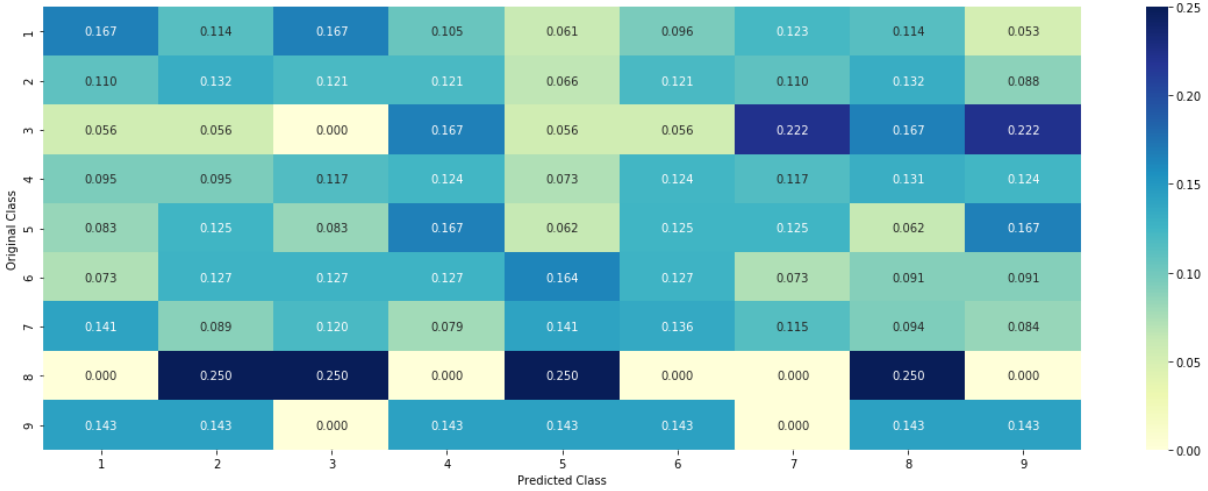
```
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_p
redicted_y, eps=1e-15))

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

```
Log loss on Cross Validation Data using Random Model 2.57316579534
Log loss on Test Data using Random Model 2.43846511559
-------------------- Confusion matrix --------------------
```




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

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



3.3 Univariate Analysis

In [22]:
```
# 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 feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_f
ea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

```
    # Amplification                             43
    # Fusions                                   22
    # Overexpression                             3
    # E17K                                       3
    # Q61L                                       3
    # S222D                                      2
    # P130S                                      2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

            # cls_cnt.shape[0](numerator) will contain the number of ti
```

```
me that particular feature occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90
*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.06818181818181818177, 0.06818181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.0606060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
    #      ...
```

```
    #     }
    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 e
ach feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if
 it is there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fe
a
    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 [23]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
```

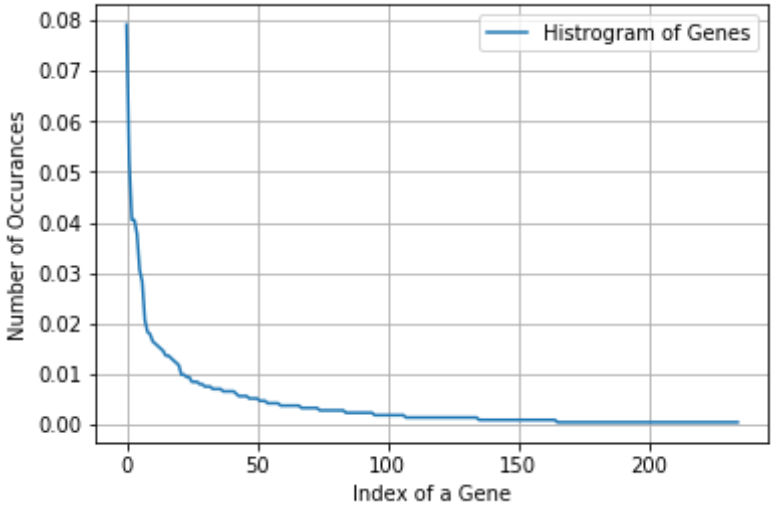
```
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 235
BRCA1     168
TP53      111
EGFR       86
PTEN       86
BRCA2      80
BRAF       65
KIT        60
ALK        44
ERBB2      39
PDGFRA     38
Name: Gene, dtype: int64
```

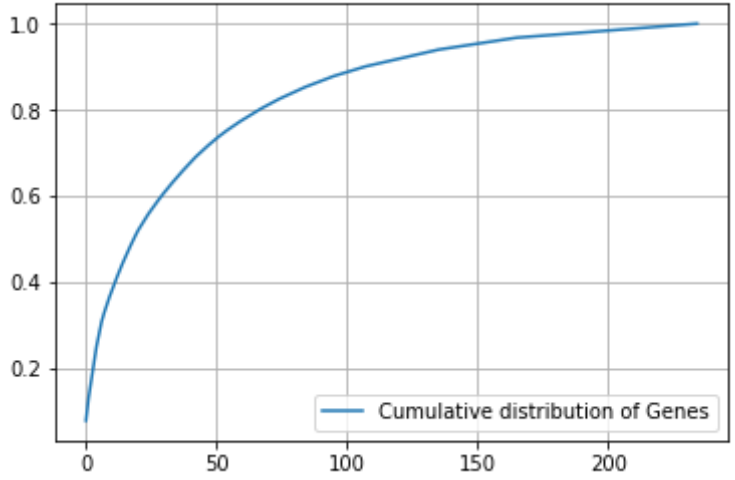
In [24]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

```
Ans: There are 235 different categories of genes in the train data, and
they are distibuted as follows
```

In [25]:
```
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 [26]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [27]:
```
#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, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [28]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [29]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
```

```
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [30]:
```
train_df['Gene'].head()
```

Out[30]:
```
364      EPAS1
927     PDGFRA
2353     AURKA
2644     BRCA1
489       TP53
Name: Gene, dtype: object
```

In [31]:
```
gene_vectorizer.get_feature_names()
```

Out[31]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
```

```
                'bard1',
                'bcl10',
                'bcl2l11',
                'bcor',
                'braf',
                'brca1',
                'brca2',
                'brd4',
                'brip1',
                'btk',
                'card11',
                'carm1',
                'casp8',
                'cbl',
                'ccnd1',
                'ccnd2',
                'ccnd3',
                'ccne1',
                'cdh1',
                'cdk12',
                'cdk4',
                'cdk6',
                'cdkn1a',
                'cdkn1b',
                'cdkn2a',
                'cdkn2b',
                'cdkn2c',
                'cebpa',
                'chek2',
                'cic',
                'crebbp',
                'ctcf',
                'ctla4',
                'ctnnb1',
                'ddr2',
                'dicer1',
                'dnmt3a',
                'dnmt3b',
                'egfr',
```

```
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'errfi1',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fanca',
                'fat1',
                'fbxw7',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxp1',
                'fubp1',
                'gata3',
                'gli1',
                'gnaq',
                'gnas',
                'h3f3a',
                'hist1h1c',
                'hla',
                'hras',
```

```
                'idh1',
                'idh2',
                'igf1r',
                'ikbke',
                'ikzf1',
                'jak1',
                'jak2',
                'jun',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
                'kmt2a',
                'kmt2b',
                'kmt2c',
                'kmt2d',
                'knstrn',
                'kras',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'mdm2',
                'mdm4',
                'med12',
                'mef2b',
                'men1',
                'met',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
```

```
                'ncor1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'notch2',
                'nras',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pax8',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pik3r3',
                'pim1',
                'pms1',
                'pms2',
                'pole',
                'ppm1d',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
                'rab35',
                'rac1',
```

```
                'rad21',
                'rad50',
                'rad51b',
                'rad51d',
                'raf1',
                'rara',
                'rasa1',
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rictor',
                'rit1',
                'rnf43',
                'ros1',
                'rras2',
                'runx1',
                'rybp',
                'sdhb',
                'sdhc',
                'setd2',
                'sf3b1',
                'shoc2',
                'smad2',
                'smad3',
                'smad4',
                'smarca4',
                'smarcb1',
                'smo',
                'sox9',
                'spop',
                'src',
                'stag2',
                'stat3',
                'stk11',
                'tcf7l2',
                'tert',
                'tet1',
```

```
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [32]:
```
# creating a pandas dataframe of the vectorized features
df_gene_train = pd.DataFrame(train_gene_feature_onehotCoding.toarray(),
 columns=gene_vectorizer.get_feature_names())
df_gene_test = pd.DataFrame(test_gene_feature_onehotCoding.toarray(), c
olumns=gene_vectorizer.get_feature_names())
df_gene_cv = pd.DataFrame(cv_gene_feature_onehotCoding.toarray(), colum
ns=gene_vectorizer.get_feature_names())
```

In [33]:
```
print("train_gene_feature_onehotCoding is converted feature using one-h
ot encoding method. The shape of gene feature:", train_gene_feature_one
hotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature: (2124, 235)
```

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

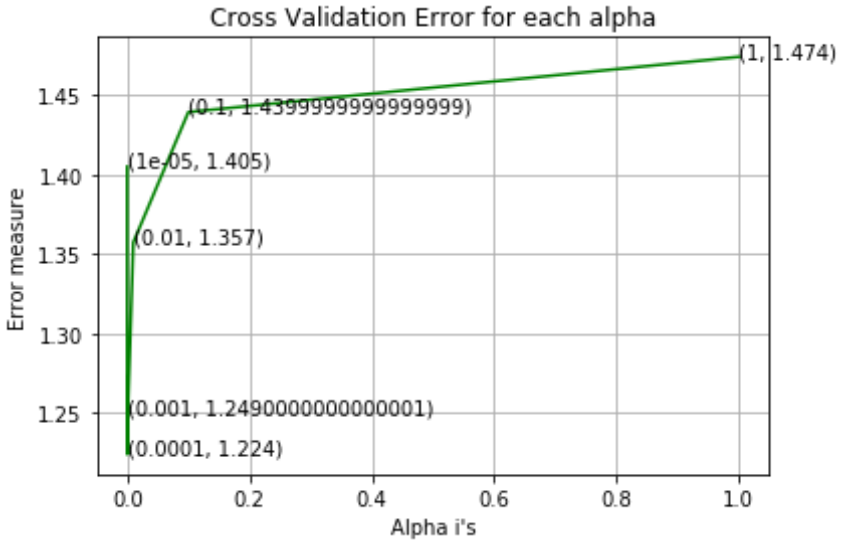
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [34]:

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

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

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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

```
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.40515960631
For values of alpha =  0.0001 The log loss is: 1.22403333621
For values of alpha =  0.001 The log loss is: 1.24949836739
For values of alpha =  0.01 The log loss is: 1.35747453599
For values of alpha =  0.1 The log loss is: 1.43952652405
For values of alpha =  1 The log loss is: 1.47433374003
```




```
For values of best alpha =  0.0001 The train log loss is: 1.02952634349
For values of best alpha =  0.0001 The cross validation log loss is: 1.
22403333621
For values of best alpha =  0.0001 The test log loss is: 1.2024352364
```

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 [35]:
```
print("Q6. How many data points in Test and CV datasets are covered by
 the ", unique_genes.shape[0], " genes in train dataset?")

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
```

```
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  23
5  genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 511 out of  532 : 96.05263157894737
```

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 [36]:
```
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))
```

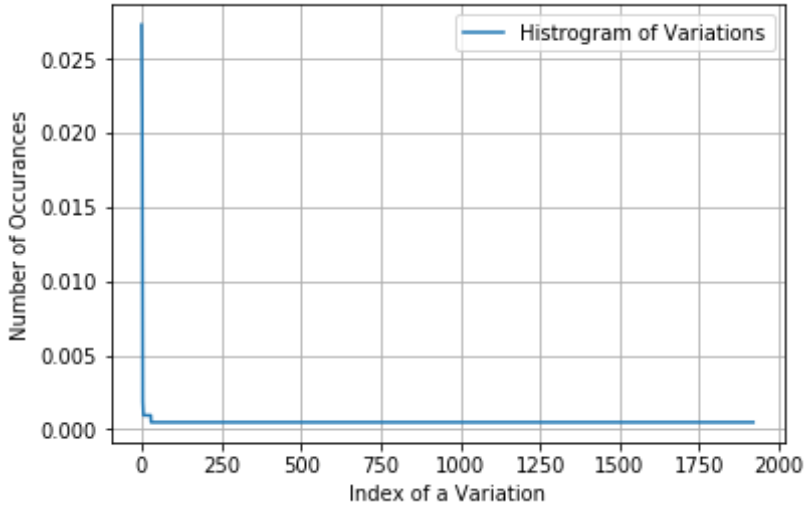
```
Number of Unique Variations : 1920
Truncating_Mutations    58
Amplification           51
Deletion                45
Fusions                 25
G12V                     4
Overexpression           3
C618R                    2
F28L                     2
G67R                     2
A146T                    2
Name: Variation, dtype: int64
```

In [37]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
```

```
,)
```

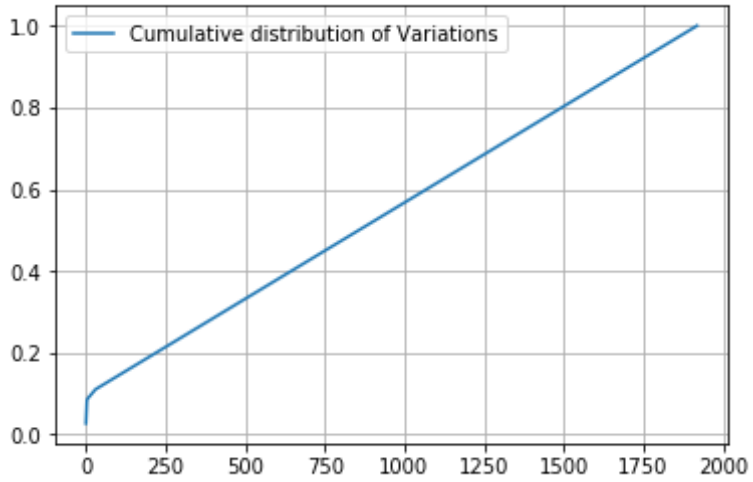
```
Ans: There are 1920 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [38]:
```
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 [39]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[ 0.02730697  0.05131827  0.07250471 ...,  0.99905838  0.99952919  1.
      ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [40]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
```

```
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [41]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [42]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [43]:
```
df_var_train = pd.DataFrame(train_variation_feature_onehotCoding.toarra
y(), columns=variation_vectorizer.get_feature_names())
df_var_test = pd.DataFrame(test_variation_feature_onehotCoding.toarray
(), columns=variation_vectorizer.get_feature_names())
df_var_cv = pd.DataFrame(cv_variation_feature_onehotCoding.toarray(), c
olumns=variation_vectorizer.get_feature_names())
```

In [44]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature: (2124, 1948)
```

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

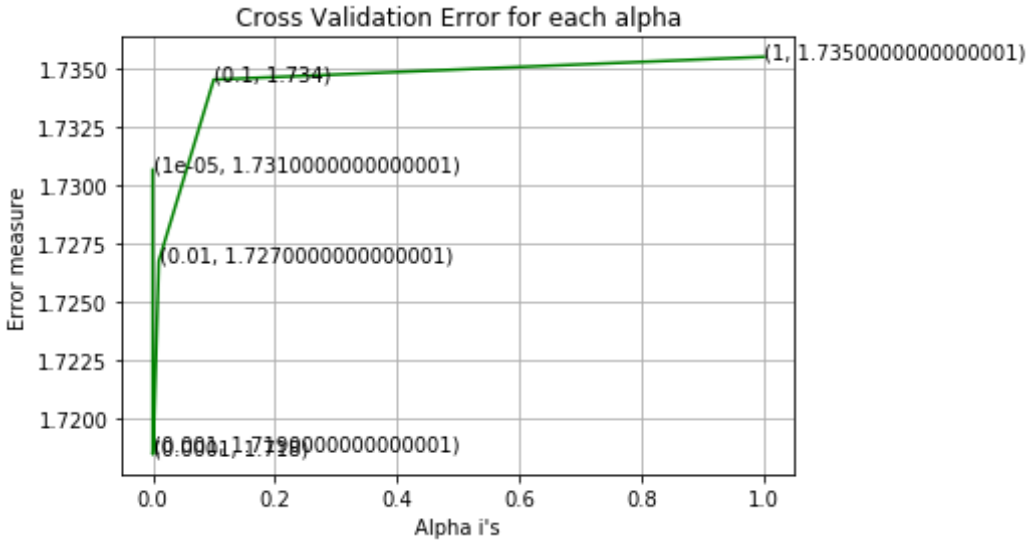
Let's build a model just like the earlier!

In [45]:

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

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

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

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


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

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

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
```

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.73062770985
For values of alpha =  0.0001 The log loss is: 1.71843634689
For values of alpha =  0.001 The log loss is: 1.71863249478
For values of alpha =  0.01 The log loss is: 1.72671614724
For values of alpha =  0.1 The log loss is: 1.73449231214
For values of alpha =  1 The log loss is: 1.73546833406
```




```
For values of best alpha =  0.0001 The train log loss is: 0.76303890186
1
For values of best alpha =  0.0001 The cross validation log loss is: 1.
71843634689
For values of best alpha =  0.0001 The test log loss is: 1.70834233769
```

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

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

In [46]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

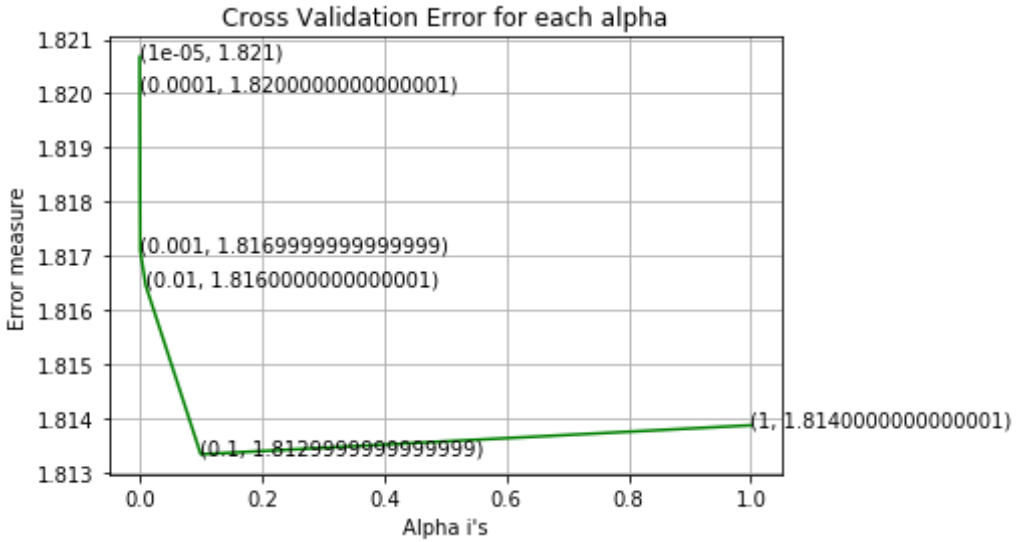
```
Q12. How many data points are covered by total  1920  genes in test and
cross validation data sets?
Ans
1. In test data 61 out of 665 : 9.172932330827068
2. In cross validation data 58 out of  532 : 10.902255639097744
```

Univariate Analysis on Word Count Feature

In [47]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_df["Word_Count"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Word_Count.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Word_Count.reshape(-1,1))

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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

```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_df.Word_Count.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Word_Count.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.Word_Count.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_df.Word_Count.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_df.Word_Count.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.82067603659
For values of alpha =  0.0001 The log loss is: 1.8200581998
For values of alpha =  0.001 The log loss is: 1.81709377436
For values of alpha =  0.01 The log loss is: 1.81645753841
For values of alpha =  0.1 The log loss is: 1.8133419143
For values of alpha =  1 The log loss is: 1.81387746001
```




```
For values of best alpha =  0.1 The train log loss is: 1.81030012554
For values of best alpha =  0.1 The cross validation log loss is: 1.813
3419143
For values of best alpha =  0.1 The test log loss is: 1.81954955266
```

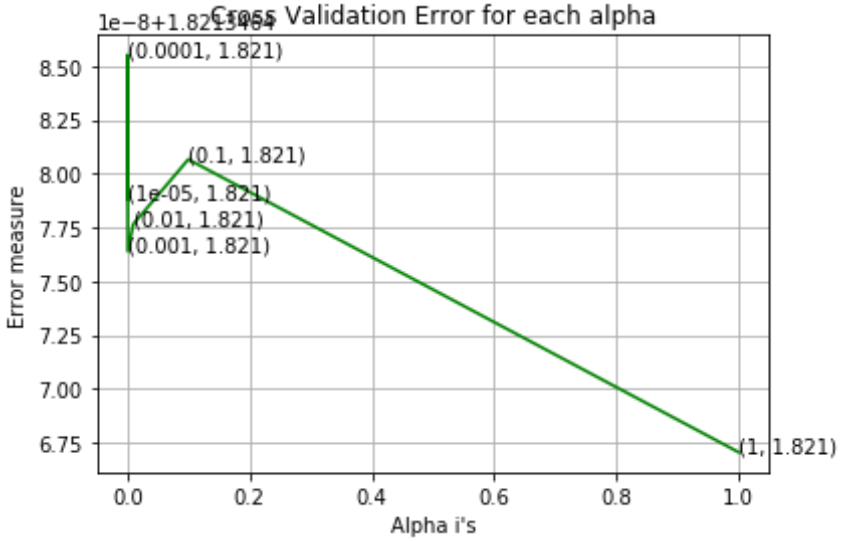
Univariate analysis on Word Count > 5000 Feature

In [48]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_df["Word_Count_5000"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Word_Count_5000.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Word_Count_5000.reshape(-1,
1))
```

```
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_df.Word_Count_5000.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Word_Count_5000.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.Word_Count_5000.reshape(-1,1
))
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_df.Word_Count_5000.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_df.Word_Count_5000.reshape(-1,1
))
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.82134647882
```

```
For values of alpha =  0.0001 The log loss is: 1.82134648552
For values of alpha =  0.001 The log loss is: 1.82134647638
For values of alpha =  0.01 The log loss is: 1.82134647765
For values of alpha =  0.1 The log loss is: 1.82134648066
For values of alpha =  1 The log loss is: 1.82134646704
```




```
For values of best alpha =  1 The train log loss is: 1.82840292277
For values of best alpha =  1 The cross validation log loss is: 1.82134
646704
For values of best alpha =  1 The test log loss is: 1.82845655979
```

Univariate Analysis on Gene and Variation Feature

In [49]:
```
genevars = train_df['Gene_Variation'].value_counts()
print('Number of Unique Gene+Variations :', genevars.shape[0])
# the top 10 variations that occured most
print(genevars.head(10))
```

```
Number of Unique Gene+Variations : 2124
PIK3CA_E542Q    1
TSC2_L493P      1
```

```
BRCA2_D1420Y    1
PIK3R1_R262T    1
BRCA2_S196N     1
ROS1_G2032R     1
CDKN2A_P81L     1
ERBB2_Y803N     1
FGFR3_H643D     1
KIT_R634W       1
Name: Gene_Variation, dtype: int64
```

In [50]:
```
# Featurizing the Gene_and_Variation Feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_and_variation_feature_responseCoding = np.array(get_gv_featu
re(alpha, "Gene_Variation", train_df))
# test gene feature
test_gene_and_variation_feature_responseCoding = np.array(get_gv_featur
e(alpha, "Gene_Variation", test_df))
# cross validation gene feature
cv_gene_and_variation_feature_responseCoding = np.array(get_gv_feature(
alpha, "Gene_Variation", cv_df))
```

In [51]:
```
# one-hot encoding of gene_and_variation feature.
gene_variation_vectorizer = CountVectorizer()
train_gene_and_variation_feature_onehotCoding = gene_variation_vectoriz
er.fit_transform(train_df["Gene_Variation"])
test_gene_and_variation_feature_onehotCoding = gene_variation_vectorize
r.transform(test_df["Gene_Variation"])
cv_gene_and_variation_feature_onehotCoding = gene_variation_vectorizer.
transform(cv_df["Gene_Variation"])
```

In [52]:
```
df_geneandvar_train = pd.DataFrame(train_gene_and_variation_feature_one
hotCoding.toarray(), columns=gene_variation_vectorizer.get_feature_name
s())
df_geneandvar_test = pd.DataFrame(test_gene_and_variation_feature_oneho
tCoding.toarray(), columns=gene_variation_vectorizer.get_feature_names
())
```

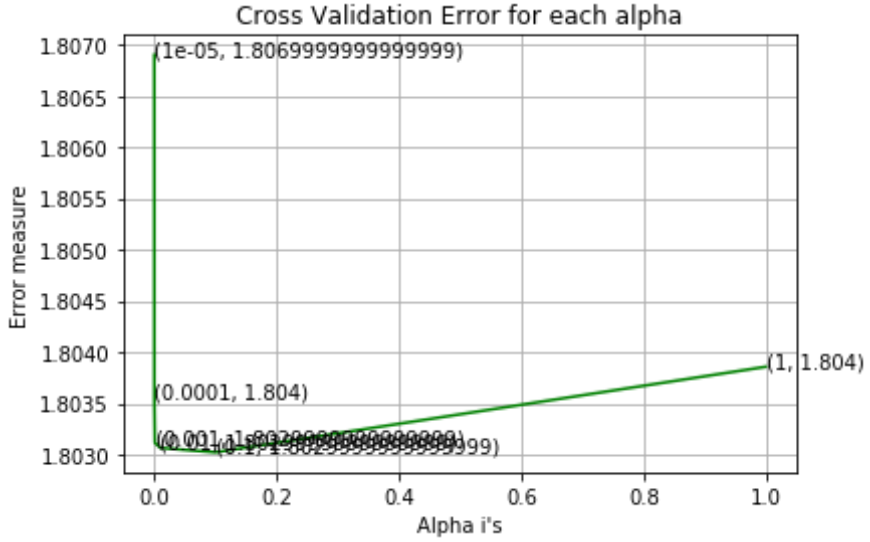
```
df_geneandvar_cv = pd.DataFrame(cv_gene_and_variation_feature_onehotCod
ing.toarray(), columns=gene_variation_vectorizer.get_feature_names())
```

In [53]:

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_geneandvar_train, y_train)
    predict_y = sig_clf.predict_proba(df_geneandvar_cv)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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


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

```
predict_y = sig_clf.predict_proba(df_geneandvar_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(df_geneandvar_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(df_geneandvar_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.80690150599
For values of alpha =  0.0001 The log loss is: 1.80354768063
For values of alpha =  0.001 The log loss is: 1.80311407301
For values of alpha =  0.01 The log loss is: 1.80306481424
For values of alpha =  0.1 The log loss is: 1.80302167034
For values of alpha =  1 The log loss is: 1.80385660762
```




```
For values of best alpha =  0.1 The train log loss is: 0.643180005585
For values of best alpha =  0.1 The cross validation log loss is: 1.803
02167034
For values of best alpha =  0.1 The test log loss is: 1.77991680377
```

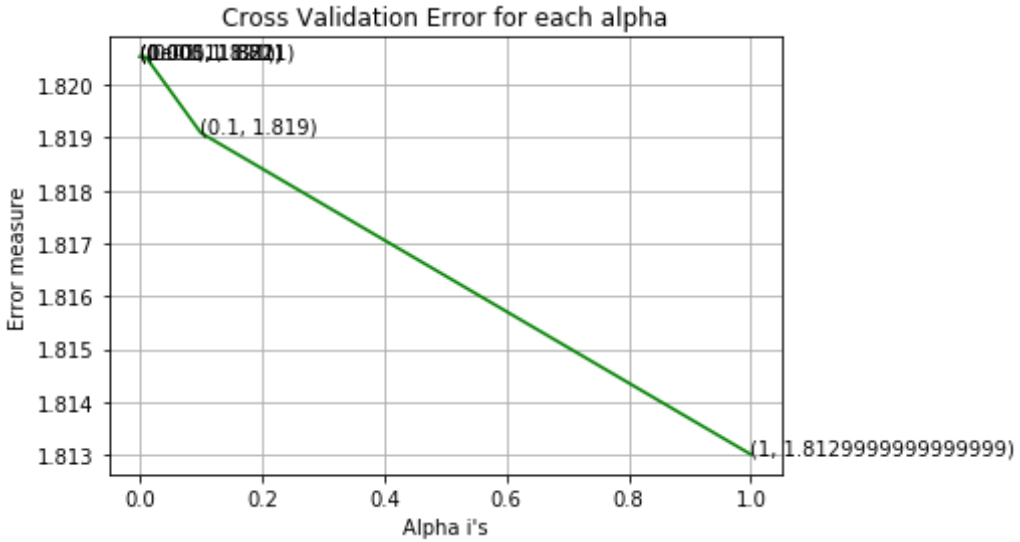
For values of best alpha = 0.1 the test log loss is: 1.7799106057

Univariate Analysis on Character Count Feature

In [54]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_df["Character_Count"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Character_Count.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Character_Count.reshape(-1,
1))

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
```

```
clf.fit(train_df.Character_Count.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Character_Count.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.Character_Count.reshape(-1,1
))
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_df.Character_Count.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_df.Character_Count.reshape(-1,1
))
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.82052654249
For values of alpha =  0.0001 The log loss is: 1.82052654249
For values of alpha =  0.001 The log loss is: 1.82052654249
For values of alpha =  0.01 The log loss is: 1.82052654249
For values of alpha =  0.1 The log loss is: 1.81909056935
For values of alpha =  1 The log loss is: 1.8130106998
```




```
For values of best alpha =  1 The train log loss is: 1.81032424458
For values of best alpha =  1 The cross validation log loss is: 1.81301
06998
For values of best alpha =  1 The test log loss is: 1.81919592071
```

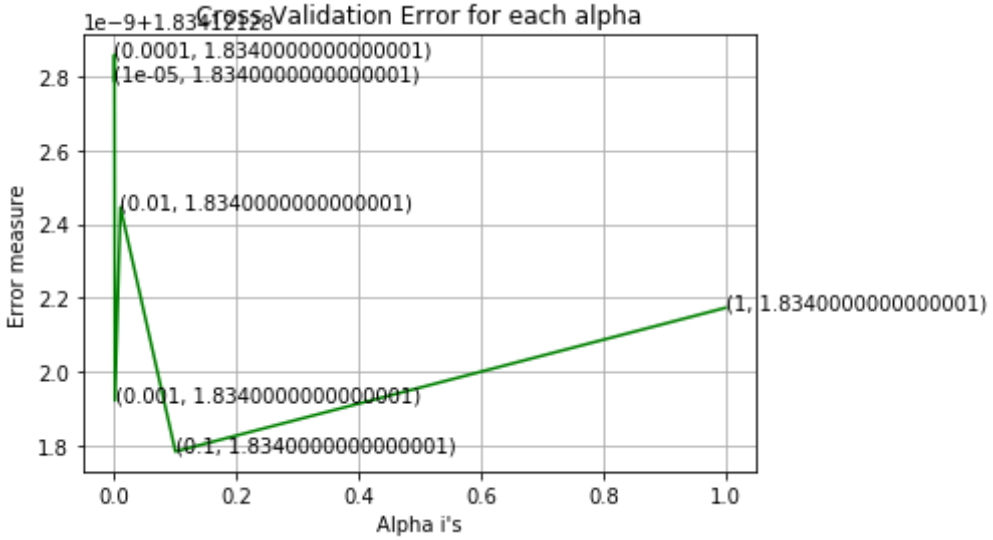
Univariate Analysis on Average Length Feature

In [55]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_df["Avg_length"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Avg_length.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Avg_length.reshape(-1,1))
```

```
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_df.Avg_length.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Avg_length.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.Avg_length.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_df.Avg_length.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_df.Avg_length.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.83412128279
For values of alpha =  0.0001 The log loss is: 1.83412128286
For values of alpha =  0.001 The log loss is: 1.83412128192
For values of alpha =  0.01 The log loss is: 1.83412128244
```

```
For values of alpha =  0.1 The log loss is: 1.83412128178
For values of alpha =  1 The log loss is: 1.83412128217
```




```
For values of best alpha =  0.1 The train log loss is: 1.82890069531
For values of best alpha =  0.1 The cross validation log loss is: 1.834
12128178
For values of best alpha =  0.1 The test log loss is: 1.82721411705
```

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 [56]:
```
# 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 [57]:
```
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 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [58]:
```
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_fea
tures=5000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

```
# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

```
Total number of unique words in train data : 5000
```

In [59]:
```
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 [60]:
```
#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 [61]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [62]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEX
T'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCodi
ng, 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)
```

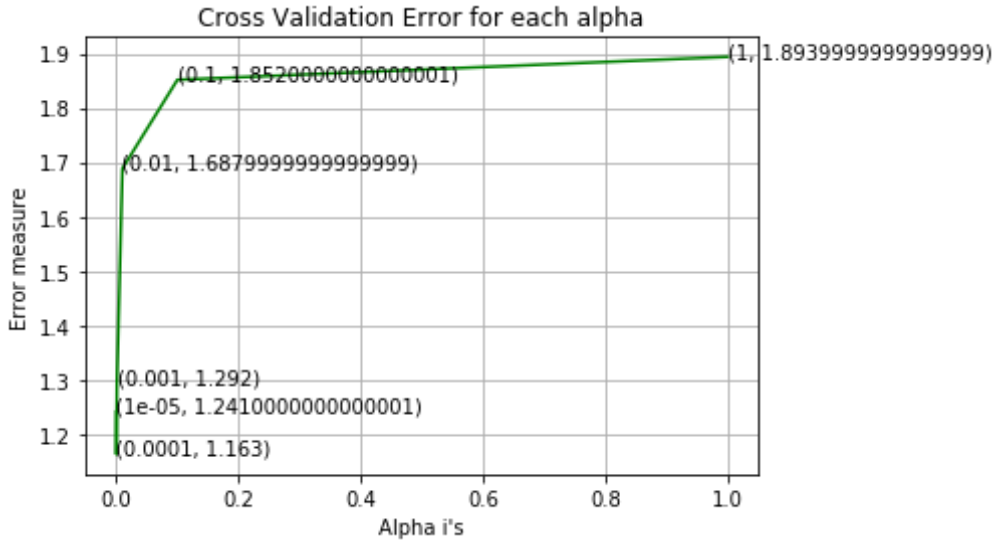
In [63]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [64]:
```
# Train a Logistic regression+Calibration model using text features whi
cha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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

```
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.24122563432
For values of alpha =  0.0001 The log loss is: 1.16299294241
For values of alpha =  0.001 The log loss is: 1.29166570786
For values of alpha =  0.01 The log loss is: 1.68841610906
For values of alpha =  0.1 The log loss is: 1.85151310874
For values of alpha =  1 The log loss is: 1.89375774441
```




```
For values of best alpha =  0.0001 The train log loss is: 0.69221606424
1
For values of best alpha =  0.0001 The cross validation log loss is: 1.
16299294241
For values of best alpha =  0.0001 The test log loss is: 1.15667372941
```

In [65]:
```
df_text_train = pd.DataFrame(train_text_feature_onehotCoding.toarray(),
 columns=text_vectorizer.get_feature_names())
df_text_test = pd.DataFrame(test_text_feature_onehotCoding.toarray(), c
olumns=text_vectorizer.get_feature_names())
df_text_cv = pd.DataFrame(cv_text_feature_onehotCoding.toarray(), colum
ns=text_vectorizer.get_feature_names())
```

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

Ans. Yes, it seems like!

In [66]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_fea
```

```
tures=5000)
    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_coun
ts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [67]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
 train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appe
ared in train data")
```

```
91.64 % of word of test data appeared in train data
90.36 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [68]:
```
#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 probabilit
ies belongs to each class
```

```
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [69]:
```
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 [70]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
 or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=10, max_features=5000, ngra
m_range=(1,4))

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

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

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

```
 [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data p
oint [{}]".format(word,yes_no))
        else:
            try:
                word = text_vec.get_feature_names()[v-(fea1_len+fea2_le
n)]
                yes_no = True if word in text.split() else False
                if yes_no:
                    word_present += 1
                    print(i, "Text feature [{}] present in test data po
int [{}]".format(word,yes_no))
            except:
                pass

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

Stacking the features

In [71]:
```
#target variables
train_y = train_df['Class'].values
test_y = test_df['Class'].values
cv_y = cv_df['Class'].values

# concatenating all the vectorized dataframes
df_gene_var_train = pd.concat([df_gene_train, df_var_train], axis=1)
df_gene_var_test = pd.concat([df_gene_test, df_var_test], axis=1)
df_gene_var_cv = pd.concat([df_gene_cv, df_var_cv], axis=1)

df_gene_and_var_train = pd.concat([df_gene_var_train, df_geneandvar_tra
```

```
in], axis=1)
df_gene_and_var_test = pd.concat([df_gene_var_test, df_geneandvar_test
], axis=1)
df_gene_and_var_cv = pd.concat([df_gene_var_cv, df_geneandvar_cv], axis
=1)

df_train = pd.concat([df_gene_and_var_train, df_text_train], axis=1)
df_test = pd.concat([df_gene_and_var_test, df_text_test], axis=1)
df_cv = pd.concat([df_gene_and_var_cv, df_text_cv], axis=1)

# scaling the text_count feature
scaler = MinMaxScaler()
train_df["Word_Count"] = scaler.fit_transform(train_df["Word_Count"].re
shape(-1,1))
test_df["Word_Count"] = scaler.fit_transform(test_df["Word_Count"].resh
ape(-1,1))
cv_df["Word_Count"] = scaler.fit_transform(cv_df["Word_Count"].reshape(
-1,1))

train_df["Character_Count"] = scaler.fit_transform(train_df["Character_
Count"].reshape(-1,1))
test_df["Character_Count"] = scaler.fit_transform(test_df["Character_Co
unt"].reshape(-1,1))
cv_df["Character_Count"] = scaler.fit_transform(cv_df["Character_Count"
].reshape(-1,1))

train_df["Avg_length"] = scaler.fit_transform(train_df["Avg_length"].re
shape(-1,1))
test_df["Avg_length"] = scaler.fit_transform(test_df["Avg_length"].resh
ape(-1,1))
cv_df["Avg_length"] = scaler.fit_transform(cv_df["Avg_length"].reshape(
-1,1))


df_train["Gene_Share"] = train_df.Gene_Share.values
df_train["Variation_Share"] = train_df.Variation_Share.values
df_train["Word_Count_5000"] = train_df.Word_Count_5000.values
df_train["Word_Count"] = train_df.Word_Count.values
df_train["Character_Count"] = train_df.Character_Count.values
```

```
df_train["Avg_length"] = train_df.Avg_length.values

df_test["Gene_Share"] = test_df.Gene_Share.values
df_test["Variation_Share"] = test_df.Variation_Share.values
df_test["Word_Count_5000"] = test_df.Word_Count_5000.values
df_test["Word_Count"] = test_df.Word_Count.values
df_test["Character_Count"] = test_df.Character_Count.values
df_test["Avg_length"] = test_df.Avg_length.values

df_cv["Gene_Share"] = cv_df.Gene_Share.values
df_cv["Variation_Share"] = cv_df.Variation_Share.values
df_cv["Word_Count_5000"] = cv_df.Word_Count_5000.values
df_cv["Word_Count"] = cv_df.Word_Count.values
df_cv["Character_Count"] = cv_df.Character_Count.values
df_cv["Avg_length"] = cv_df.Avg_length.values

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
cv_variation_feature_responseCoding))

train_geneandvar_responseCoding = np.hstack((train_gene_var_responseCod
ing,train_gene_and_variation_feature_responseCoding))
test_geneandvar_responseCoding = np.hstack((test_gene_var_responseCodin
g,test_gene_and_variation_feature_responseCoding))
cv_geneandvar_responseCoding = np.hstack((cv_gene_var_responseCoding,cv
_gene_and_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_geneandvar_responseCoding, tr
ain_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_geneandvar_responseCoding, test
_text_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_geneandvar_responseCoding, cv_text_
feature_responseCoding))

train_geneshare = np.column_stack((train_x_responseCoding, train_df.Gen
e_Share.values))
```

```
test_geneshare = np.column_stack((test_x_responseCoding, test_df.Gene_S
hare.values))
cv_geneshare = np.column_stack((cv_x_responseCoding, cv_df.Gene_Share.v
alues))

train_varshare = np.column_stack((train_geneshare, train_df.Variation_S
hare.values))
test_varshare = np.column_stack((test_geneshare, test_df.Variation_Shar
e.values))
cv_varshare = np.column_stack((cv_geneshare, cv_df.Variation_Share.valu
es))

train_text5000 = np.column_stack((train_varshare, train_df.Word_Count_5
000.values))
test_text5000 = np.column_stack((test_varshare, test_df.Word_Count_5000
.values))
cv_text5000 = np.column_stack((cv_varshare, cv_df.Word_Count_5000.value
s))

train_x_response = np.column_stack((train_text5000, train_df.Word_Count
.values))
test_x_response = np.column_stack((test_text5000, test_df.Word_Count.va
lues))
cv_x_response = np.column_stack((cv_text5000, cv_df.Word_Count.values))



train_x_response = np.column_stack((train_x_response, train_df.Characte
r_Count.values))
test_x_response = np.column_stack((test_x_response, test_df.Character_C
ount.values))
cv_x_response = np.column_stack((cv_x_response, cv_df.Character_Count.v
alues))

train_x_response = np.column_stack((train_x_response, train_df.Avg_leng
th.values))
test_x_response = np.column_stack((test_x_response, test_df.Avg_length.
values))
cv_x_response = np.column_stack((cv_x_response, cv_df.Avg_length.values
```

```
))



train_x_onehotCoding = df_train
test_x_onehotCoding = df_test
cv_x_onehotCoding = df_cv

train_x_responseCoding = train_x_response
test_x_responseCoding = test_x_response
cv_x_responseCoding = cv_x_response
```

In [72]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 93
56)
(number of data points * number of features) in test data =  (665, 935
6)
(number of data points * number of features) in cross validation data =
(532, 9356)
```

In [73]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 4
2)
```

```
(number of data points * number of features) in test data =  (665, 42)
(number of data points * number of features) in cross validation data =
(532, 42)
```

4.1. Base Line Model

4.1.1. Naive Bayes

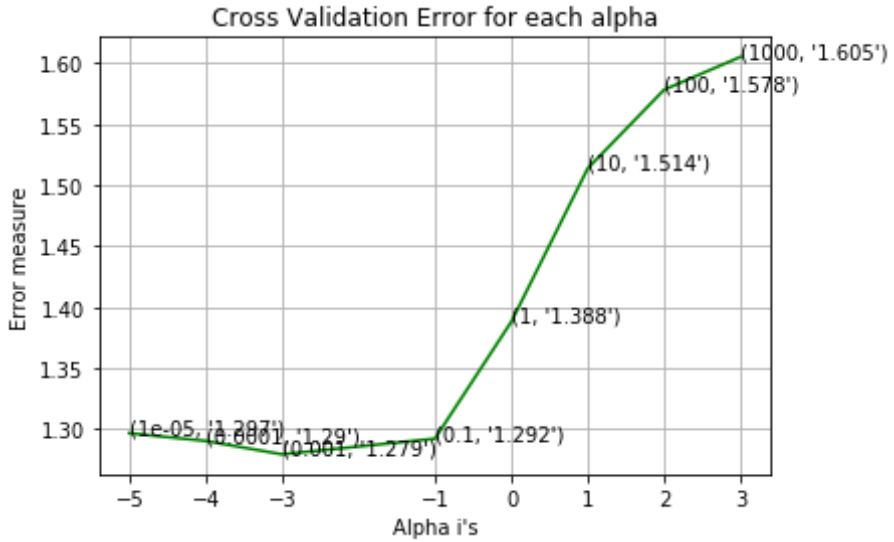
4.1.1.1. Hyper parameter tuning

In [74]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=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 v
ector X.
# ----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/naive-bayes-algorithm-1/
# ----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
```

```
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/naive-bayes-algorithm-1/
# ----------------------


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

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

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


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

```
for alpha = 1e-05
Log Loss : 1.296710263
for alpha = 0.0001
Log Loss : 1.29040191011
for alpha = 0.001
Log Loss : 1.27942602284
for alpha = 0.1
Log Loss : 1.29220378393
for alpha = 1
Log Loss : 1.38806546589
for alpha = 10
Log Loss : 1.51377356332
for alpha = 100
Log Loss : 1.5782148899
for alpha = 1000
Log Loss : 1.60519098505
```




```
For values of best alpha =  0.001 The train log loss is: 0.665728436824
For values of best alpha =  0.001 The cross validation log loss is: 1.2
7942602284
For values of best alpha =  0.001 The test log loss is: 1.28930983392
```

4.1.1.2. Testing the model with best hyper paramters

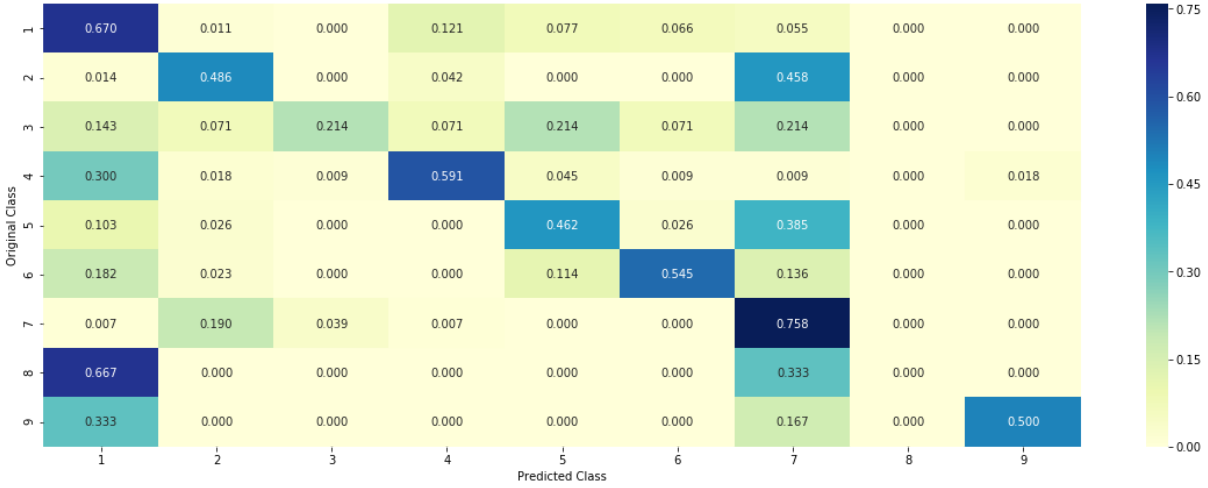
In [75]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

# 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 v
ector X.
# ----------------------
```

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


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

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

```
Log Loss : 1.27942602284
Number of missclassified point : 0.3890977443609023
-------------------- Confusion matrix --------------------
```

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

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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9

4.1.1.3. Feature Importance, Correctly classified point

In [76]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[ 0.0612  0.0505  0.0106  0.079   0.033
9  0.6507  0.1071  0.0044  0.0025]]
Actual Class : 6
--------------------------------------------------
36 Text feature [ring] present in test data point [True]
52 Text feature [noted] present in test data point [True]
58 Text feature [white] present in test data point [True]
78 Text feature [important] present in test data point [True]
79 Text feature [middle] present in test data point [True]
87 Text feature [however] present in test data point [True]
Out of the top  100  features  6 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [77]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.5456  0.0608  0.0127  0.1644  0.040
7  0.0388  0.1288  0.0053  0.003 ]]
Actual Class : 1
--------------------------------------------------
16 Text feature [probability] present in test data point [True]
```

```
26 Text feature [mean] present in test data point [True]
30 Text feature [well] present in test data point [True]
41 Text feature [isoforms] present in test data point [True]
47 Text feature [three] present in test data point [True]
55 Text feature [indicated] present in test data point [True]
71 Text feature [top] present in test data point [True]
82 Text feature [removal] present in test data point [True]
86 Text feature [invasive] present in test data point [True]
92 Text feature [usually] present in test data point [True]
96 Text feature [variable] present in test data point [True]
Out of the top  100  features  11 are present in query point
```

4.2. K Nearest Neighbour Classification

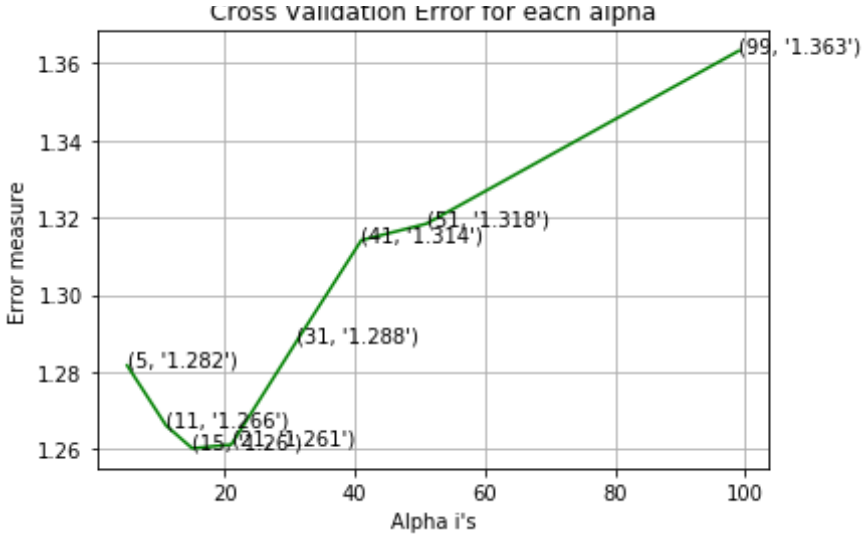
4.2.1. Hyper parameter tuning

In [78]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

```
for alpha = 5
Log Loss : 1.28170673022
for alpha = 11
Log Loss : 1.26611535807
for alpha = 15
Log Loss : 1.26026310918
for alpha = 21
Log Loss : 1.26121475488
for alpha = 31
Log Loss : 1.28785313111
for alpha = 41
Log Loss : 1.31411304755
for alpha = 51
Log Loss : 1.31837823038
for alpha = 99
Log Loss : 1.36311345732
```




```
For values of best alpha =  15 The train log loss is: 0.980363525505
For values of best alpha =  15 The cross validation log loss is: 1.2602
6310918
For values of best alpha =  15 The test log loss is: 1.20113890867
```

4.2.2. Testing the model with best hyper paramters

In [79]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

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

```
Log loss : 1.26026310918
Number of mis-classified points : 0.4492481203007519
-------------------- Confusion matrix --------------------
```


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

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

Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9
K Nearest	Response Coding	0.9803	1.2602	1.2011	44.9

Neighbours					

4.2.3.Sample Query point -1

In [80]:
```
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].resh
ape(1, -1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points
 belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 6
The  15  nearest neighbours of the test points belongs to classes [6 6
6 6 6 6 6 6 6 6 6 6 6 6]
Fequency of nearest points : Counter({6: 15})
```

4.2.4. Sample Query Point-2

In [81]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
```

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

```
Predicted Class : 1
Actual Class : 1
the k value for knn is 15 and the nearest neighbours of the test points
belongs to classes [1 1 1 1 1 1 4 1 1 1 1 1 1 1 1]
Fequency of nearest points : Counter({1: 14, 4: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

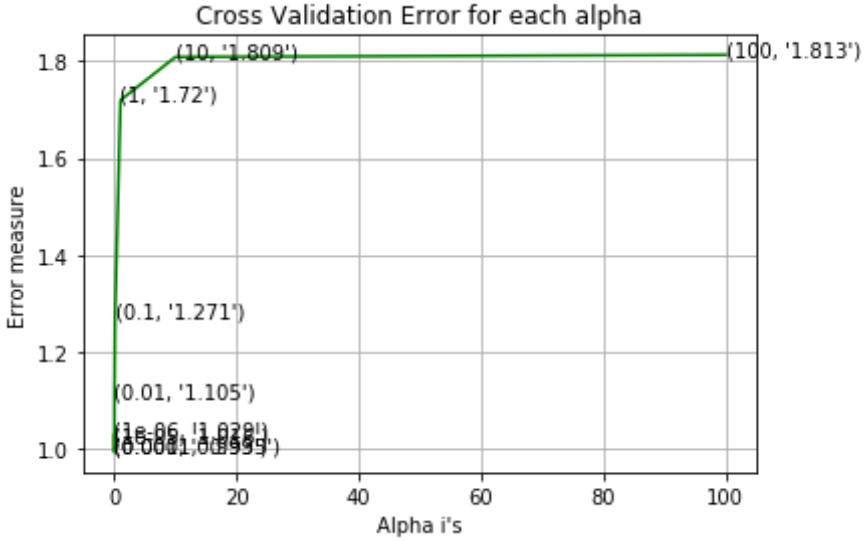
In [82]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))
```

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


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

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

```
for alpha = 1e-06
Log Loss : 1.02868924079
for alpha = 1e-05
Log Loss : 1.01588973997
for alpha = 0.0001
Log Loss : 0.994946210035
for alpha = 0.001

Log Loss : 0.992981843829
```

```
for alpha = 0.01
Log Loss : 1.10484154219
for alpha = 0.1
Log Loss : 1.27070908848
for alpha = 1
Log Loss : 1.71954514545
for alpha = 10
Log Loss : 1.80870919071
for alpha = 100
Log Loss : 1.81311359071
```



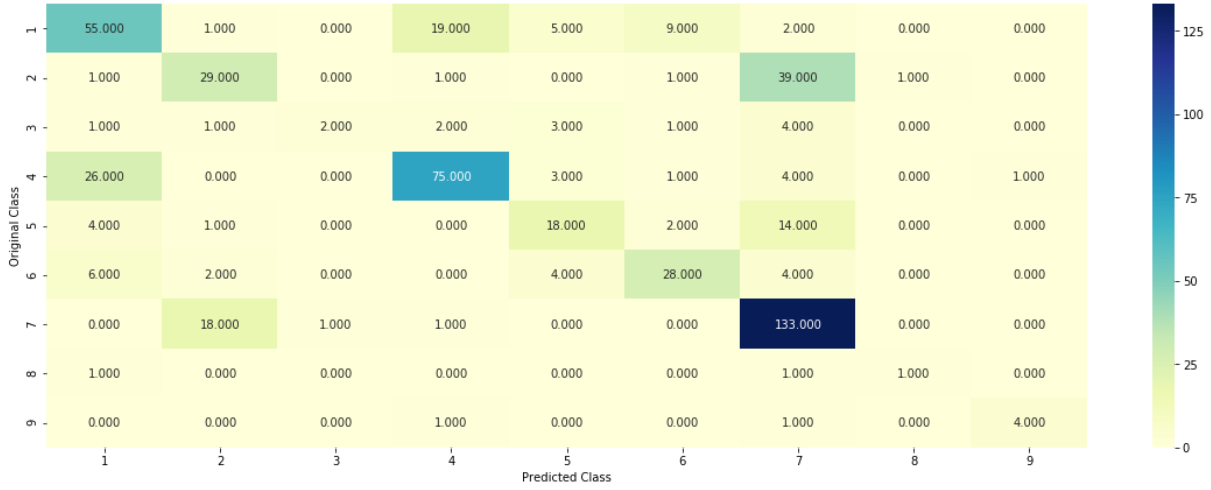

```
For values of best alpha =  0.001 The train log loss is: 0.540484800282
For values of best alpha =  0.001 The cross validation log loss is: 0.9
92981843829
For values of best alpha =  0.001 The test log loss is: 1.00432865876
```

4.3.1.2. Testing the model with best hyper paramters

In [83]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
```

```
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

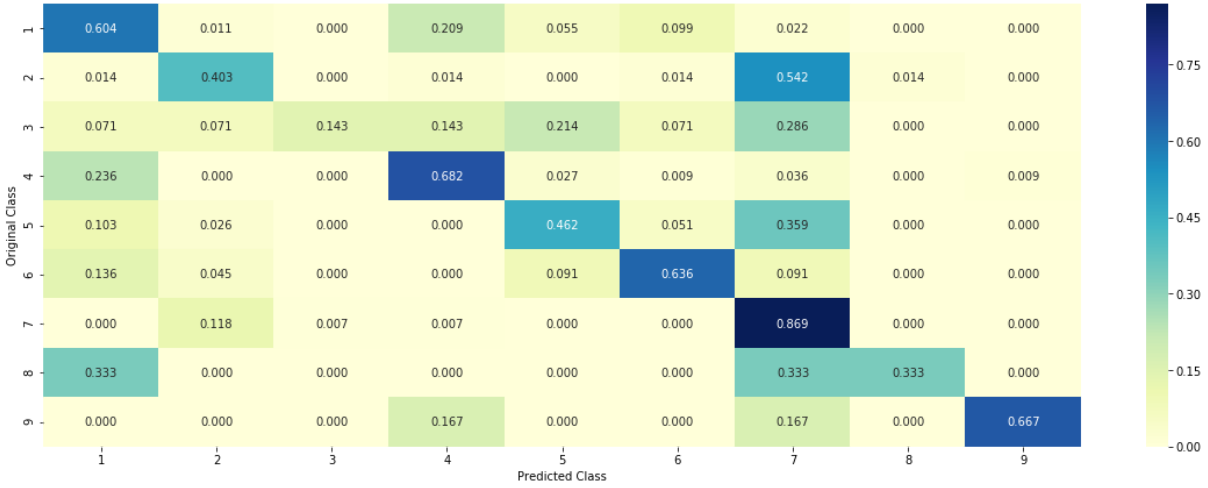
```
Log loss : 0.992981843829
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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

--

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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9

K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15

4.3.1.3. Feature Importance

In [84]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
ry point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

4.3.1.3.1. Correctly Classified point

In [85]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[  8.30000000e-03   5.20000000e-03   1.
90000000e-03   3.90000000e-03
    1.20400000e-01   8.58100000e-01   3.00000000e-04   1.30000000e-03
    6.00000000e-04]]
Actual Class : 6
--------------------------------------------------
11 Text feature [32] present in test data point [True]
15 Text feature [access] present in test data point [True]
20 Text feature [components] present in test data point [True]
39 Text feature [cross] present in test data point [True]
65 Text feature [4a] present in test data point [True]
72 Text feature [coding] present in test data point [True]
78 Text feature [71] present in test data point [True]
82 Text feature [complex] present in test data point [True]
90 Text feature [carcinoma] present in test data point [True]
107 Text feature [category] present in test data point [True]
118 Text feature [containing] present in test data point [True]
120 Text feature [assumed] present in test data point [True]
152 Text feature [two] present in test data point [True]
168 Text feature [origin] present in test data point [True]
```

```
169 Text feature [proteins] present in test data point [True]
189 Text feature [according] present in test data point [True]
191 Text feature [75] present in test data point [True]
204 Text feature [subgroups] present in test data point [True]
260 Text feature [brca2] present in test data point [True]
295 Text feature [align] present in test data point [True]
301 Text feature [receiving] present in test data point [True]
310 Text feature [process] present in test data point [True]
313 Text feature [repair] present in test data point [True]
317 Text feature [practice] present in test data point [True]
325 Text feature [clearly] present in test data point [True]
328 Text feature [numbers] present in test data point [True]
365 Text feature [western] present in test data point [True]
383 Text feature [likelihood] present in test data point [True]
385 Text feature [subject] present in test data point [True]
400 Text feature [groups] present in test data point [True]
407 Text feature [level] present in test data point [True]
418 Text feature [noted] present in test data point [True]
419 Text feature [proposed] present in test data point [True]
420 Text feature [original] present in test data point [True]
434 Text feature [increased] present in test data point [True]
442 Text feature [nearly] present in test data point [True]
451 Text feature [pattern] present in test data point [True]
453 Text feature [reasons] present in test data point [True]
455 Text feature [three] present in test data point [True]
459 Text feature [white] present in test data point [True]
471 Text feature [unclear] present in test data point [True]
473 Text feature [missense] present in test data point [True]
475 Text feature [ring] present in test data point [True]
483 Text feature [a1708e] present in test data point [True]
488 Text feature [table] present in test data point [True]
Out of the top  500  features  45 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [86]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
```

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

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.6188  0.0213  0.006   0.3182  0.013
5  0.0161  0.0032  0.0015  0.0013]]
Actual Class : 1
--------------------------------------------------
21 Text feature [mutation] present in test data point [True]
33 Text feature [targets] present in test data point [True]
39 Text feature [involvement] present in test data point [True]
51 Text feature [peak] present in test data point [True]
55 Text feature [encompassing] present in test data point [True]
72 Text feature [developmental] present in test data point [True]
94 Text feature [death] present in test data point [True]
98 Text feature [acting] present in test data point [True]
134 Text feature [displays] present in test data point [True]
142 Text feature [62] present in test data point [True]
164 Text feature [cancers] present in test data point [True]
170 Text feature [proteasomal] present in test data point [True]
177 Text feature [subgroup] present in test data point [True]
197 Text feature [could] present in test data point [True]
208 Text feature [exhibit] present in test data point [True]
259 Text feature [case] present in test data point [True]
295 Text feature [dysregulation] present in test data point [True]
306 Text feature [established] present in test data point [True]
310 Text feature [95] present in test data point [True]
332 Text feature [table] present in test data point [True]
357 Text feature [families] present in test data point [True]
370 Text feature [abstract] present in test data point [True]
373 Text feature [well] present in test data point [True]
384 Text feature [pdgf] present in test data point [True]
```

```
387 Text feature [probes] present in test data point [True]
392 Text feature [failed] present in test data point [True]
401 Text feature [germany] present in test data point [True]
408 Text feature [surgical] present in test data point [True]
417 Text feature [diagnosis] present in test data point [True]
423 Text feature [partial] present in test data point [True]
436 Text feature [analysis] present in test data point [True]
440 Text feature [addition] present in test data point [True]
457 Text feature [spots] present in test data point [True]
469 Text feature [additional] present in test data point [True]
478 Text feature [88] present in test data point [True]
495 Text feature [aspects] present in test data point [True]
Out of the top  500  features  36 are present in query point
```

4.3.2. Without Class balancing

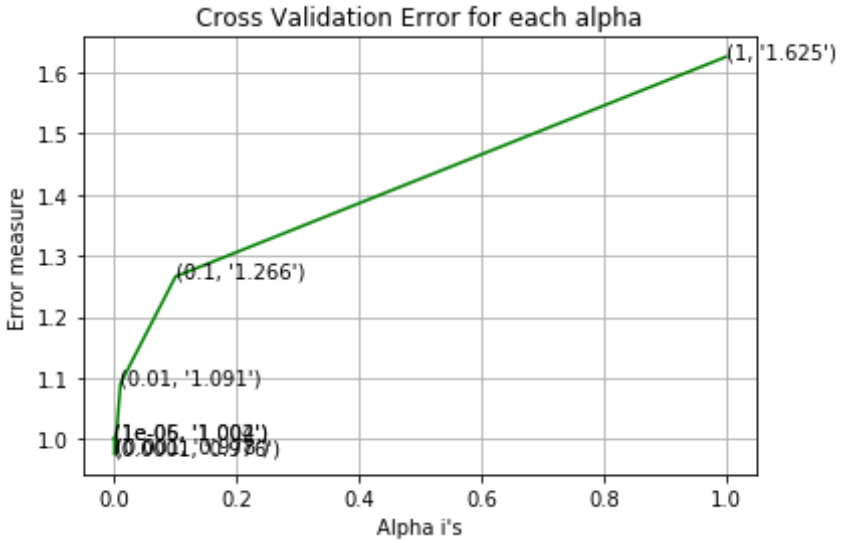
4.3.2.1. Hyper paramter tuning

In [87]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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



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

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.00196014424
for alpha = 1e-05
Log Loss : 1.00392867124
for alpha = 0.0001
Log Loss : 0.975520924748
for alpha = 0.001
Log Loss : 0.978012610236
for alpha = 0.01
Log Loss : 1.09139347577
for alpha = 0.1
Log Loss : 1.26611001688
for alpha = 1
Log Loss : 1.62523857562
```




```
For values of best alpha =  0.0001 The train log loss is: 0.41579130920
7
For values of best alpha =  0.0001 The cross validation log loss is: 0.
975520924748
For values of best alpha =  0.0001 The test log loss is: 0.993878897581
```

4.3.2.2. Testing model with best hyper parameters

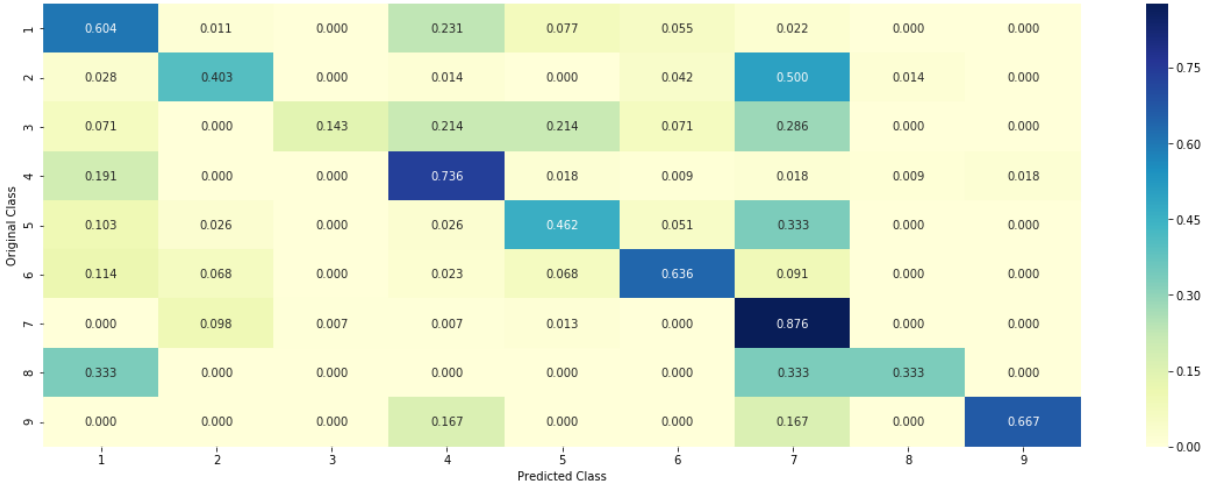
In [88]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with S
```

```
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

```
Log loss : 0.975520924748
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```


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

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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9

K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15
Logistics Regression without Class Balancing	**OneHotEncoded**	**0.4157**	**0.9755**	**0.9938**	**33.83**

4.3.2.3. Feature Importance, Correctly Classified point

In [89]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[  1.89000000e-02   9.00000000e-03   3.
60000000e-03   1.48000000e-02
    7.54000000e-02   8.74600000e-01   9.00000000e-04   2.10000000e-03
    6.00000000e-04]]
Actual Class : 6
--------------------------------------------------
0 Text feature [access] present in test data point [True]
4 Text feature [32] present in test data point [True]
```

```
10 Text feature [cross] present in test data point [True]
34 Text feature [complex] present in test data point [True]
38 Text feature [71] present in test data point [True]
45 Text feature [carcinoma] present in test data point [True]
53 Text feature [components] present in test data point [True]
66 Text feature [containing] present in test data point [True]
67 Text feature [4a] present in test data point [True]
76 Text feature [coding] present in test data point [True]
78 Text feature [category] present in test data point [True]
121 Text feature [according] present in test data point [True]
165 Text feature [75] present in test data point [True]
189 Text feature [align] present in test data point [True]
204 Text feature [assumed] present in test data point [True]
211 Text feature [origin] present in test data point [True]
216 Text feature [receiving] present in test data point [True]
219 Text feature [practice] present in test data point [True]
223 Text feature [brca2] present in test data point [True]
231 Text feature [two] present in test data point [True]
250 Text feature [clearly] present in test data point [True]
266 Text feature [a1708e] present in test data point [True]
284 Text feature [repair] present in test data point [True]
335 Text feature [pattern] present in test data point [True]
351 Text feature [proteins] present in test data point [True]
354 Text feature [level] present in test data point [True]
355 Text feature [western] present in test data point [True]
363 Text feature [alleles] present in test data point [True]
378 Text feature [reasons] present in test data point [True]
392 Text feature [subgroups] present in test data point [True]
395 Text feature [process] present in test data point [True]
403 Text feature [proposed] present in test data point [True]
412 Text feature [unclear] present in test data point [True]
419 Text feature [subject] present in test data point [True]
437 Text feature [located] present in test data point [True]
461 Text feature [nearly] present in test data point [True]
484 Text feature [original] present in test data point [True]
Out of the top  500  features  37 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [90]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.531   0.0195  0.0081  0.394   0.018
9  0.0179  0.0057  0.0031  0.0017]]
Actual Class : 1
--------------------------------------------------
28 Text feature [acting] present in test data point [True]
37 Text feature [developmental] present in test data point [True]
64 Text feature [62] present in test data point [True]
77 Text feature [diagnosis] present in test data point [True]
79 Text feature [encompassing] present in test data point [True]
100 Text feature [abstract] present in test data point [True]
114 Text feature [displays] present in test data point [True]
116 Text feature [additional] present in test data point [True]
121 Text feature [88] present in test data point [True]
127 Text feature [cancers] present in test data point [True]
149 Text feature [germany] present in test data point [True]
163 Text feature [death] present in test data point [True]
170 Text feature [95] present in test data point [True]
179 Text feature [exhibit] present in test data point [True]
187 Text feature [could] present in test data point [True]
205 Text feature [dysregulation] present in test data point [True]
209 Text feature [foci] present in test data point [True]
226 Text feature [failed] present in test data point [True]
257 Text feature [case] present in test data point [True]
269 Text feature [73] present in test data point [True]
```

```
282 Text feature [mutation] present in test data point [True]
283 Text feature [generally] present in test data point [True]
311 Text feature [relatives] present in test data point [True]
322 Text feature [established] present in test data point [True]
341 Text feature [peak] present in test data point [True]
342 Text feature [targets] present in test data point [True]
349 Text feature [side] present in test data point [True]
350 Text feature [figure] present in test data point [True]
360 Text feature [abnormal] present in test data point [True]
403 Text feature [addition] present in test data point [True]
416 Text feature [spots] present in test data point [True]
440 Text feature [proteasomal] present in test data point [True]
471 Text feature [focus] present in test data point [True]
493 Text feature [involvement] present in test data point [True]
Out of the top  500  features  34 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

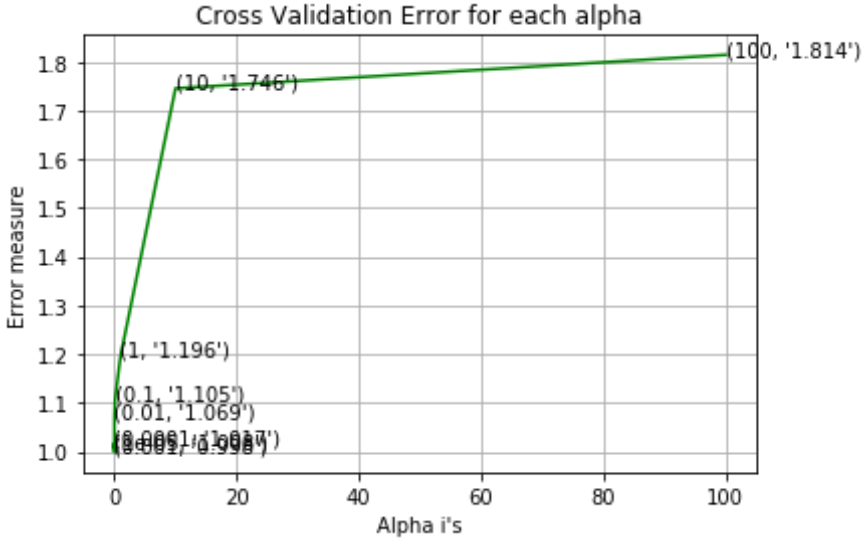
In [91]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

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

```
online/lessons/mathematical-derivation-copy-8/
# ------------------------------



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

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

fig, ax = plt.subplots()
```

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


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

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

```
for C = 1e-05
Log Loss : 1.00834355198
for C = 0.0001
Log Loss : 1.01737963952
for C = 0.001
Log Loss : 0.998433738303
for C = 0.01
Log Loss : 1.06851421541
for C = 0.1
```

```
Log Loss : 1.10545050859
for C = 1
Log Loss : 1.19589202418
for C = 10
Log Loss : 1.74563434091
for C = 100
Log Loss : 1.81446166784
```



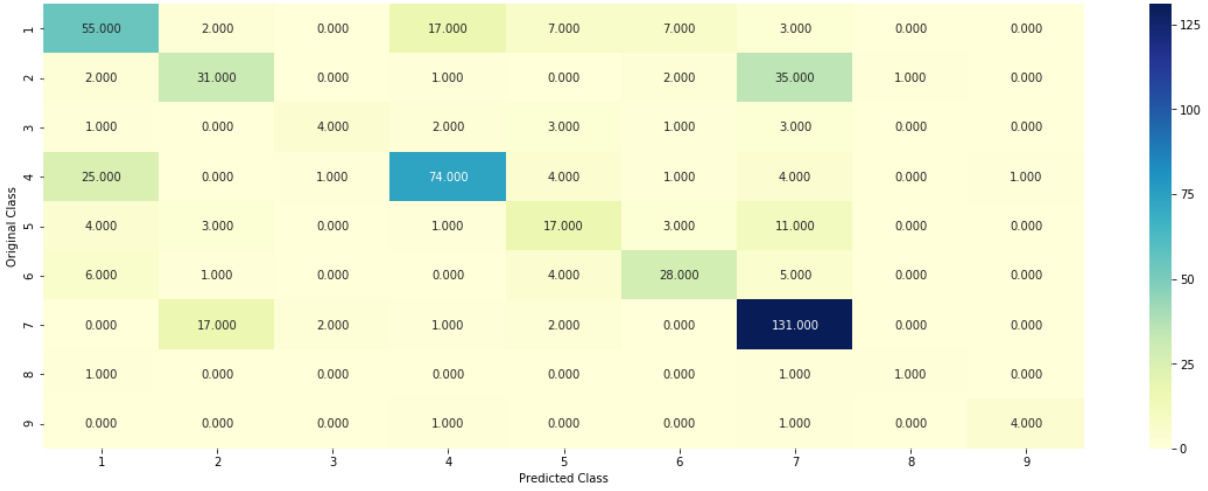

```
For values of best alpha =  0.001 The train log loss is: 0.475240998914
For values of best alpha =  0.001 The cross validation log loss is: 0.9
98433738303
For values of best alpha =  0.001 The test log loss is: 1.03478232902
```

4.4.2. Testing model with best hyper parameters

In [92]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
```

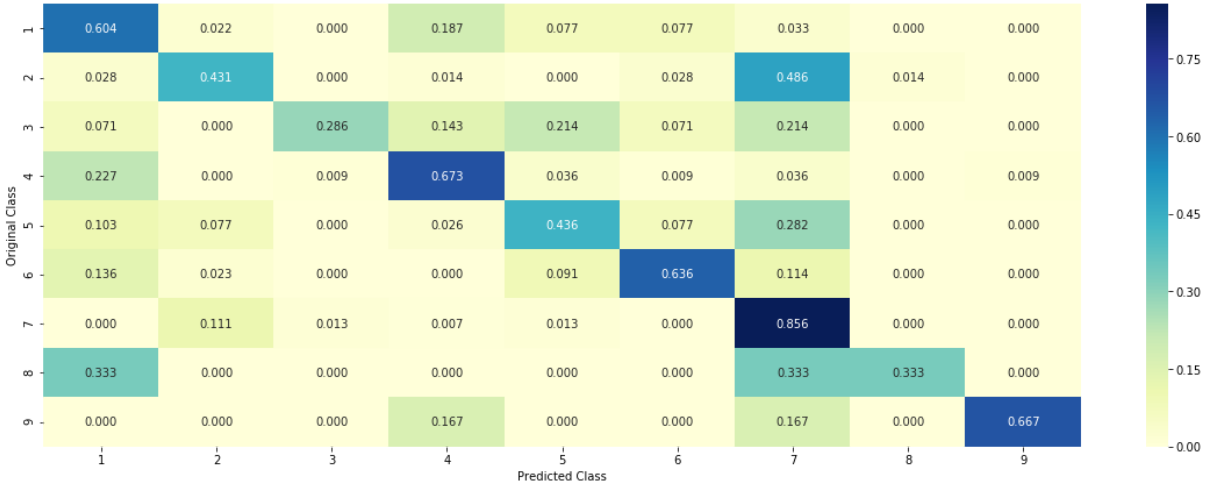
```
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

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


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

```
Log loss : 0.998433738303
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```



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


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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points

Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9
K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15
Logistics Regression without Class Balancing	**OneHotEncoded**	**0.4157**	**0.9755**	**0.9938**	**33.83**
Linear Support Vector Machines	OneHotEncoded	0.4752	0.9984	1.034	35.15

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [93]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[ 0.0168  0.0281  0.0023  0.0223  0.054
```

```
Predicted class Probabilities: [[ 0.0168  0.0281  0.0023  0.0223  0.034
5  0.8684  0.0054  0.0012  0.0011]]
Actual Class : 6
--------------------------------------------------
8 Text feature [71] present in test data point [True]
11 Text feature [components] present in test data point [True]
21 Text feature [complex] present in test data point [True]
31 Text feature [access] present in test data point [True]
46 Text feature [4a] present in test data point [True]
76 Text feature [32] present in test data point [True]
79 Text feature [category] present in test data point [True]
103 Text feature [carcinoma] present in test data point [True]
106 Text feature [containing] present in test data point [True]
110 Text feature [cross] present in test data point [True]
125 Text feature [align] present in test data point [True]
129 Text feature [brca2] present in test data point [True]
137 Text feature [assumed] present in test data point [True]
148 Text feature [according] present in test data point [True]
164 Text feature [coding] present in test data point [True]
166 Text feature [75] present in test data point [True]
205 Text feature [origin] present in test data point [True]
253 Text feature [proteins] present in test data point [True]
269 Text feature [subgroups] present in test data point [True]
273 Text feature [two] present in test data point [True]
279 Text feature [practice] present in test data point [True]
282 Text feature [receiving] present in test data point [True]
291 Text feature [repair] present in test data point [True]
324 Text feature [unclear] present in test data point [True]
355 Text feature [process] present in test data point [True]
362 Text feature [al] present in test data point [True]
368 Text feature [affected] present in test data point [True]
407 Text feature [a1708e] present in test data point [True]
409 Text feature [clearly] present in test data point [True]
412 Text feature [84] present in test data point [True]
422 Text feature [proposed] present in test data point [True]
428 Text feature [groups] present in test data point [True]
451 Text feature [subject] present in test data point [True]
455 Text feature [preliminary] present in test data point [True]
472 Text feature [likelihood] present in test data point [True]

484 Text feature [subtle] present in test data point [True]
```

```
484 Text feature [subtle] present in test data point [True]
485 Text feature [reasons] present in test data point [True]
493 Text feature [pattern] present in test data point [True]
496 Text feature [includes] present in test data point [True]
499 Text feature [well] present in test data point [True]
Out of the top  500  features  40 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [94]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.626   0.0239  0.0072  0.2925  0.015
6  0.0237  0.0076  0.0016  0.0018]]
Actual Class : 1
--------------------------------------------------
29 Text feature [95] present in test data point [True]
35 Text feature [62] present in test data point [True]
54 Text feature [developmental] present in test data point [True]
57 Text feature [73] present in test data point [True]
92 Text feature [addition] present in test data point [True]
109 Text feature [encompassing] present in test data point [True]
127 Text feature [could] present in test data point [True]
141 Text feature [acting] present in test data point [True]
145 Text feature [death] present in test data point [True]
160 Text feature [mutation] present in test data point [True]
175 Text feature [diagnosis] present in test data point [True]
```

```
205 Text feature [families] present in test data point [True]
222 Text feature [analysis] present in test data point [True]
234 Text feature [88] present in test data point [True]
239 Text feature [germany] present in test data point [True]
269 Text feature [additional] present in test data point [True]
280 Text feature [displays] present in test data point [True]
285 Text feature [full] present in test data point [True]
294 Text feature [dysregulation] present in test data point [True]
295 Text feature [abnormal] present in test data point [True]
301 Text feature [case] present in test data point [True]
311 Text feature [abstract] present in test data point [True]
329 Text feature [exhibit] present in test data point [True]
345 Text feature [cancers] present in test data point [True]
354 Text feature [targets] present in test data point [True]
367 Text feature [peak] present in test data point [True]
381 Text feature [proteasomal] present in test data point [True]
388 Text feature [spots] present in test data point [True]
396 Text feature [side] present in test data point [True]
400 Text feature [involvement] present in test data point [True]
402 Text feature [relatives] present in test data point [True]
411 Text feature [substitutions] present in test data point [True]
417 Text feature [generally] present in test data point [True]
420 Text feature [functional] present in test data point [True]
446 Text feature [finding] present in test data point [True]
447 Text feature [arise] present in test data point [True]
458 Text feature [fa] present in test data point [True]
Out of the top  500  features  37 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [95]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
```

```
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
```

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.19878478098
for n_estimators = 100 and max depth =  10
Log Loss : 1.1751071122
for n_estimators = 200 and max depth =  5
Log Loss : 1.19399464471
for n_estimators = 200 and max depth =  10
Log Loss : 1.15475197326
for n_estimators = 500 and max depth =  5
Log Loss : 1.18413743902
for n_estimators = 500 and max depth =  10
Log Loss : 1.15062233055
for n_estimators = 1000 and max depth =  5
Log Loss : 1.17713662526
for n_estimators = 1000 and max depth =  10
Log Loss : 1.14729074575
for n_estimators = 2000 and max depth =  5
Log Loss : 1.17606149786
for n_estimators = 2000 and max depth =  10
Log Loss : 1.14469162204
For values of best estimator =  2000 The train log loss is: 0.578907944
683
For values of best estimator =  2000 The cross validation log loss is:
1.14469540699
For values of best estimator =  2000 The test log loss is: 1.1411368891
4
```

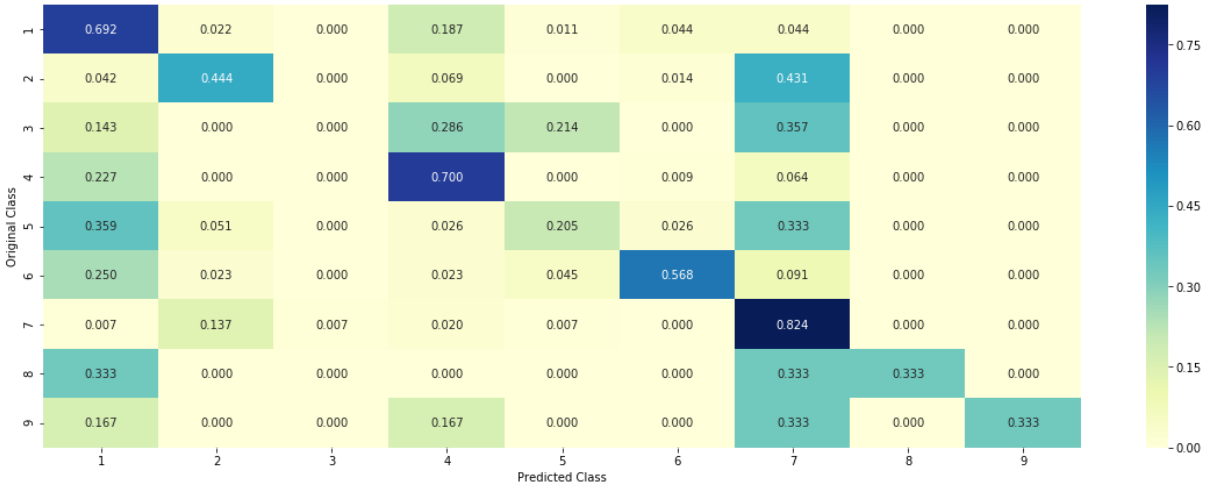
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [96]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

```
Log loss : 1.14469422174
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

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

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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9
K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15
Logistics Regression without Class Balancing	**OneHotEncoded**	**0.4157**	**0.9755**	**0.9938**	**33.83**
Linear Support Vector Machines	OneHotEncoded	0.4752	0.9984	1.034	35.15
Random Forest Classifier	OneHotEncoded	0.5789	1.4469	1.1411	37.21

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [97]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 6
Predicted Class Probabilities: [[ 0.0304  0.0174  0.0113  0.0208  0.202
5  0.6915  0.0188  0.0027  0.0046]]
Actual Class : 6
--------------------------------------------------
18 Text feature [well] present in test data point [True]
23 Text feature [statistical] present in test data point [True]
45 Text feature [tolerated] present in test data point [True]
69 Text feature [similar] present in test data point [True]
86 Text feature [respectively] present in test data point [True]
89 Text feature [probably] present in test data point [True]
99 Text feature [many] present in test data point [True]
Out of the top  100  features  7 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [98]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_ind
ex, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1)),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.3477  0.1433  0.0211  0.2526  0.060
9  0.0516  0.1052  0.0074  0.0102]]
Actuall Class : 1
--------------------------------------------------
18 Text feature [well] present in test data point [True]
34 Text feature [mutants] present in test data point [True]
53 Text feature [prostate] present in test data point [True]
69 Text feature [similar] present in test data point [True]
75 Text feature [retinoblastoma] present in test data point [True]
79 Text feature [rearrangements] present in test data point [True]
80 Text feature [retention] present in test data point [True]
86 Text feature [respectively] present in test data point [True]
88 Text feature [interval] present in test data point [True]
99 Text feature [many] present in test data point [True]
Out of the top  100  features  10 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [99]:
```
# -------------------------------
```

```
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

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

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

```
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tra
in log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cro
ss validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.87476467811
for n_estimators = 10 and max depth =  3
Log Loss : 2.59123418654
for n_estimators = 10 and max depth =  5
Log Loss : 2.52496769682
for n_estimators = 10 and max depth =  10
Log Loss : 2.06951280986
for n_estimators = 50 and max depth =  2
Log Loss : 2.78397280124
for n_estimators = 50 and max depth =  3
Log Loss : 2.24182199673
for n_estimators = 50 and max depth =  5
Log Loss : 2.29364992536
for n_estimators = 50 and max depth =  10
Log Loss : 2.42753666951
for n_estimators = 100 and max depth =  2
Log Loss : 2.47977065034
for n_estimators = 100 and max depth =  3
Log Loss : 2.21842587346
for n_estimators = 100 and max depth =  5
Log Loss : 2.10105684273
for n_estimators = 100 and max depth =  10
Log Loss : 2.35777045293
for n_estimators = 200 and max depth =  2
```

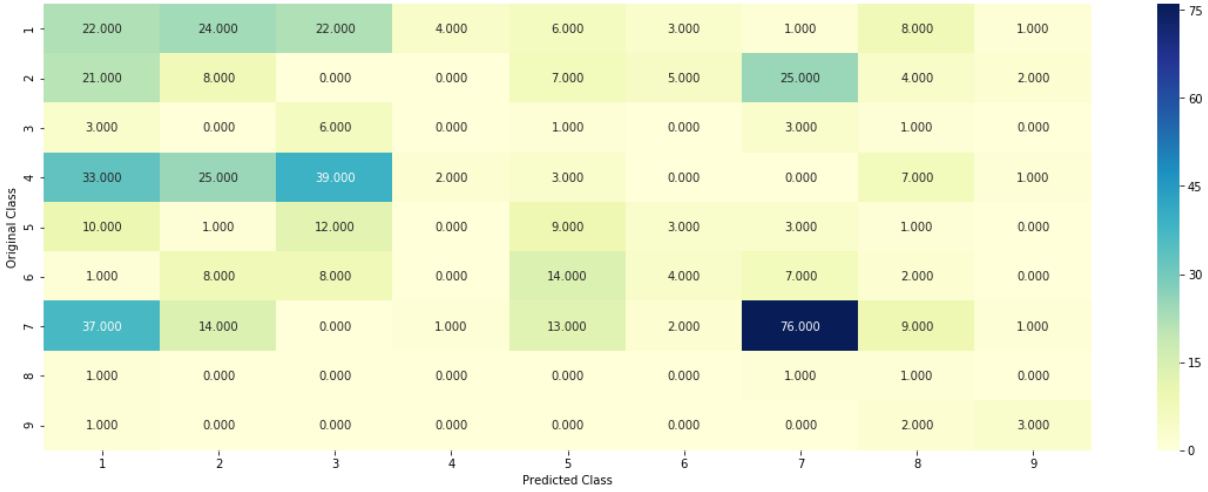
```
Log Loss : 2.48474491008

for n_estimators = 200 and max depth =  3
Log Loss : 2.33356062118
for n_estimators = 200 and max depth =  5
Log Loss : 2.09463395903
for n_estimators = 200 and max depth =  10
Log Loss : 2.31031721154
for n_estimators = 500 and max depth =  2
Log Loss : 2.31035902072
for n_estimators = 500 and max depth =  3
Log Loss : 2.21746061283
for n_estimators = 500 and max depth =  5
Log Loss : 2.078868771
for n_estimators = 500 and max depth =  10
Log Loss : 2.32841225456
for n_estimators = 1000 and max depth =  2
Log Loss : 2.27149720457
for n_estimators = 1000 and max depth =  3
Log Loss : 2.19902214856
for n_estimators = 1000 and max depth =  5
Log Loss : 2.07568377799
for n_estimators = 1000 and max depth =  10
Log Loss : 2.31647138872
For values of best alpha =  10 The train log loss is: 0.0155346298389
For values of best alpha =  10 The cross validation log loss is: 2.0695
1280986
For values of best alpha =  10 The test log loss is: 2.07407190856
```

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

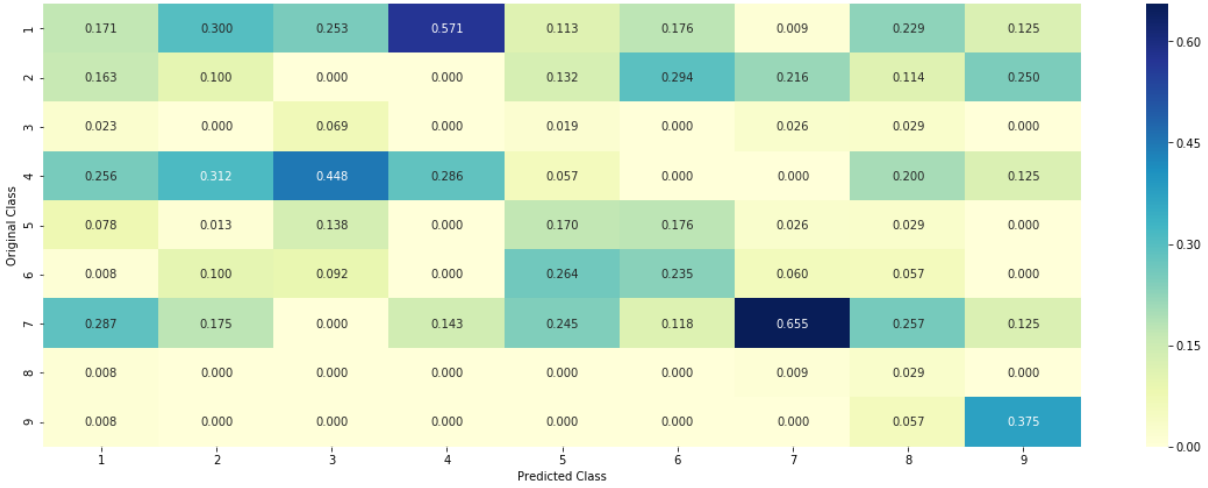
In [100]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
```

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

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

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

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='au
to',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_
responseCoding,cv_y, clf)
```

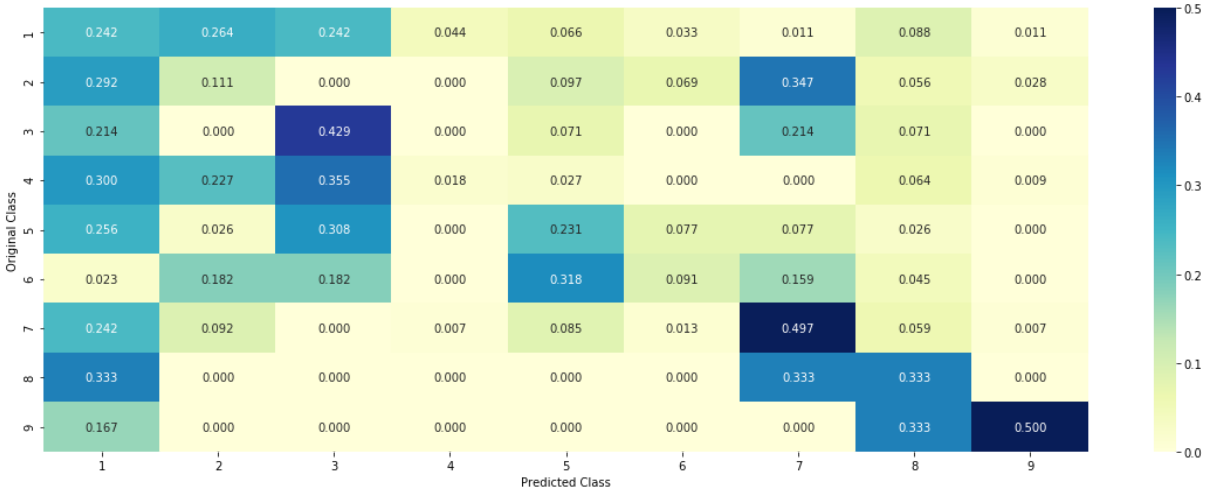
```
Log loss : 2.06951280986
Number of mis-classified points : 0.7537593984962406
-------------------- Confusion matrix --------------------
```



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



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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9
K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15
Logistics Regression without Class Balancing	**OneHotEncoded**	**0.4157**	**0.9755**	**0.9938**	**33.83**
Linear Support Vector Machines	OneHotEncoded	0.4752	0.9984	1.034	35.15
Random Forest Classifier	OneHotEncoded	0.5789	1.4469	1.1411	37.21
Random Forest Classifier	Response	0.0155	2.0695	2.0740	75.37

	Coding				

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [101]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 5
Predicted Class Probabilities: [[ 0.046   0.094   0.2307  0.0546  0.290
4  0.0744  0.0482  0.0686  0.093 ]]
Actual Class : 6
```

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

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

4.5.5.2. Incorrectly Classified point

In [102]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.6838  0.0116  0.0103  0.2706  0.007
6  0.0056  0.0042  0.0034  0.003 ]]
Actual Class : 1
--------------------------------------------------
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [103]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFo
restClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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

```
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_cl
f1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig
_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predic
t_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3
], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

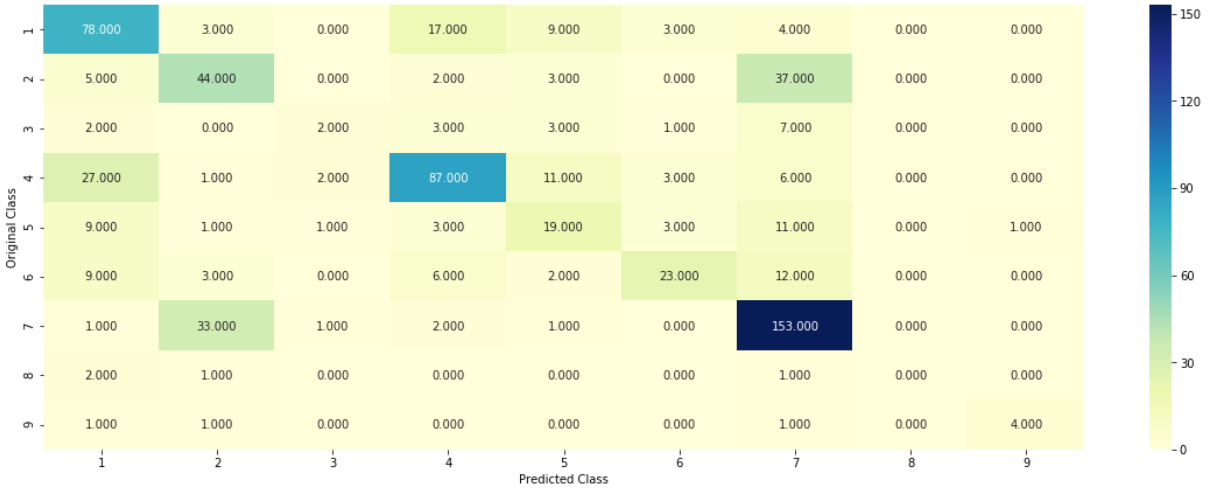
```
Logistic Regression :  Log Loss: 0.99
Support vector machines : Log Loss: 1.16
Naive Bayes : Log Loss: 1.28
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.176
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.015
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.456
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.123

Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.352
```

```
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.951
```

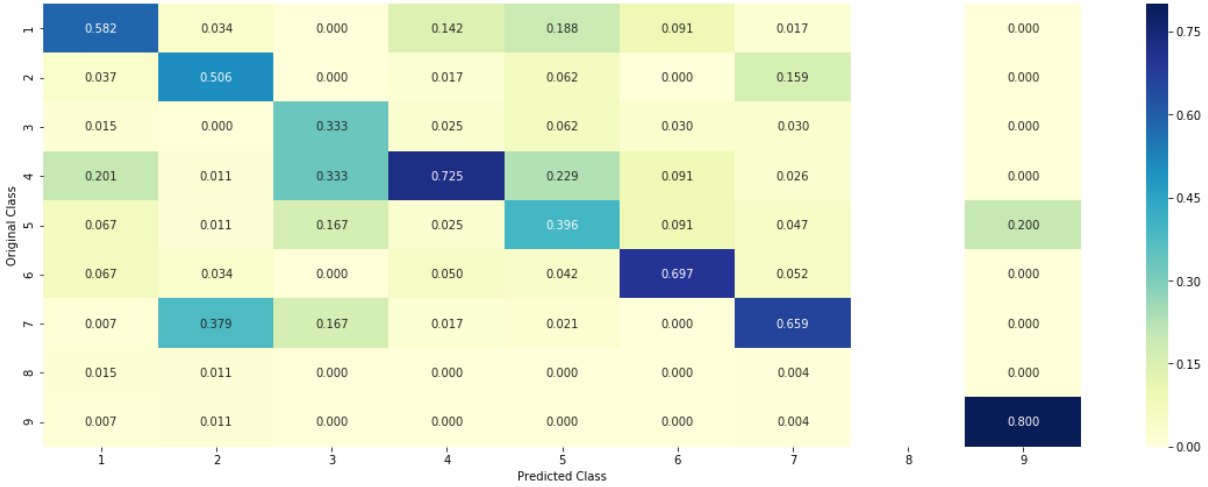
4.7.2 testing the model with the best hyper parameters

In [104]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

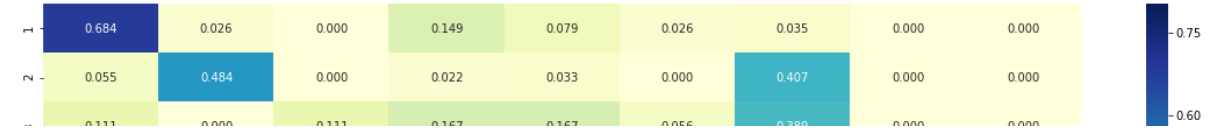
```
Log loss (train) on the stacking classifier : 0.529017353651
Log loss (CV) on the stacking classifier : 1.12271883274
Log loss (test) on the stacking classifier : 1.14095360311
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```

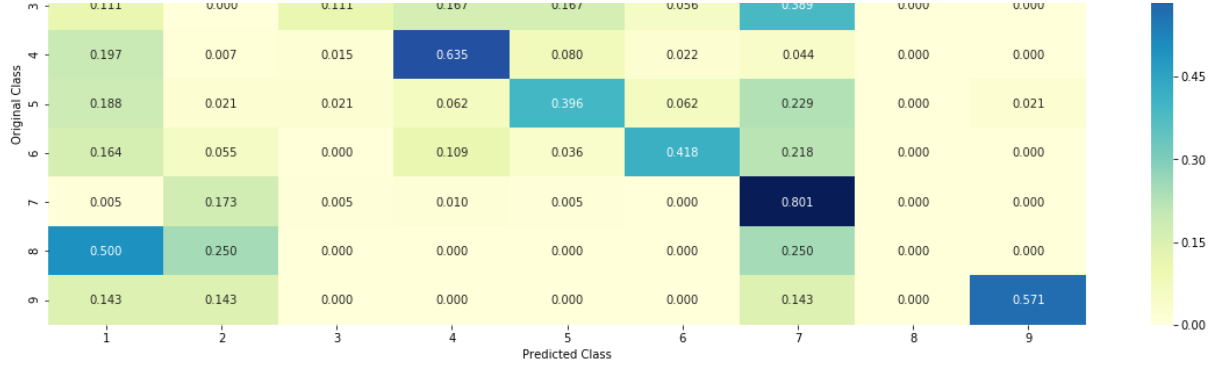


---------- Precision matrix (Columm Sum=1) ----------



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





Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9
K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15
Logistics Regression without Class Balancing	**OneHotEncoded**	**0.4157**	**0.9755**	**0.9938**	**33.83**
Linear Support Vector Machines	OneHotEncoded	0.4752	0.9984	1.034	35.15
Random Forest Classifier	OneHotEncoded	0.5789	1.4469	1.1411	37.21
Random Forest Classifier	Response Coding	0.0155	2.0695	2.0740	75.37
Stacking Classifier	OneHotEncoded	0.5290	1.1227	1.1409	38.34

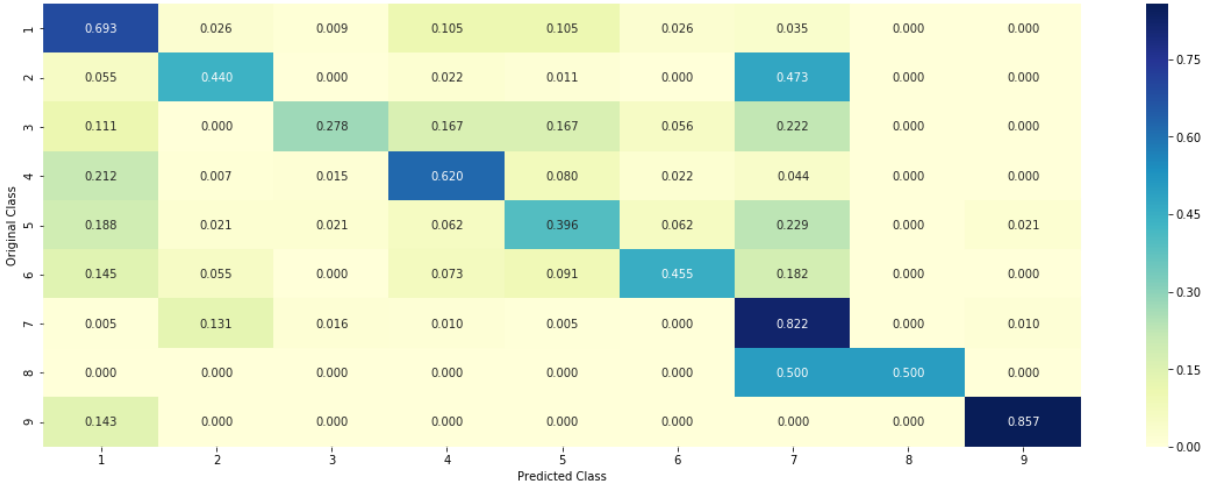
4.7.3 Maximum Voting classifier

In [105]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.652506446531
Log loss (CV) on the VotingClassifier : 1.09196579921
Log loss (test) on the VotingClassifier : 1.08468113395
Number of missclassified point : 0.37142857142857144
-------------------- Confusion matrix --------------------
```


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

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



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9

K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEcnoded	0.5404	0.9929	1.0043	35.15
Logistics Regression without Class Balancing	**OneHotEncoded**	**0.4157**	**0.9755**	**0.9938**	**33.83**
Linear Support Vector Machines	OneHotEncoded	0.4752	0.9984	1.034	35.15
Random Forest Classifier	OneHotEncoded	0.5789	1.4469	1.1411	37.21
Random Forest Classifier	Response Coding	0.0155	2.0695	2.0740	75.37
Stacking Classifier	OneHotEncoded	0.5290	1.1227	1.1409	38.34
Voting Classifier	OneHotEncoded	0.6525	1.0919	1.0846	37.14

Results -

1. Logistics Regression obtained the best result using OneHotEncoded Features.
2. Random Forest obtained the worst result using Response Coded Features.