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 [3]:
```
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.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: Dep
recationWarning: The module is deprecated in version 0.21 and will be r
emoved in version 0.23 since we've dropped support for Python 2.7. Plea
se rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [4]:
```
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[4]:

	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 [5]:
```
# 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[5]:

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

3.1.3. Preprocessing of text

In [6]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[6]: True

In [0]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [8]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_tim
e, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 373.418321 seconds
```

In [9]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[9]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...

	ID	Gene	Variation	Class	TEXT
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

In [10]:
```
result[result.isnull().any(axis=1)]
```

Out[10]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

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

In [12]:
```
result[result['ID']==1109]
```

Out[12]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [0]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution 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)
# 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)
```

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

In [0]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
])
```

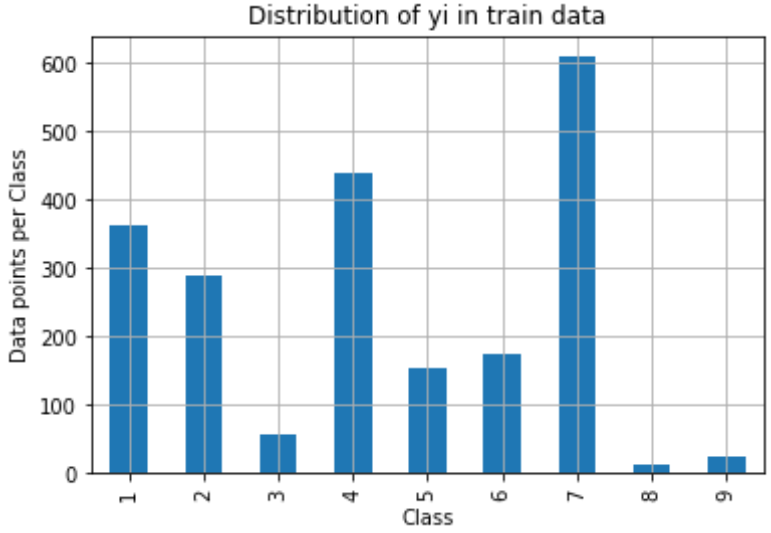
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

In [0]:
```
# 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().sort_index
()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
```

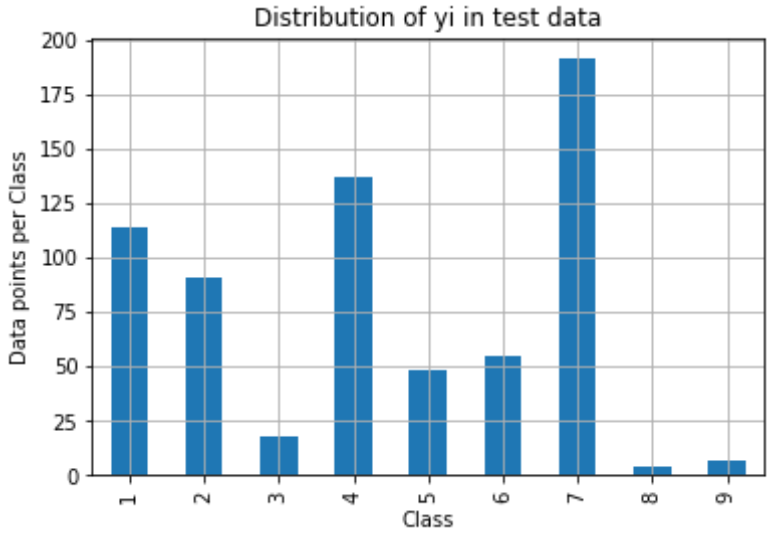
```
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# 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')
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')
```

```
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), '%)')
```

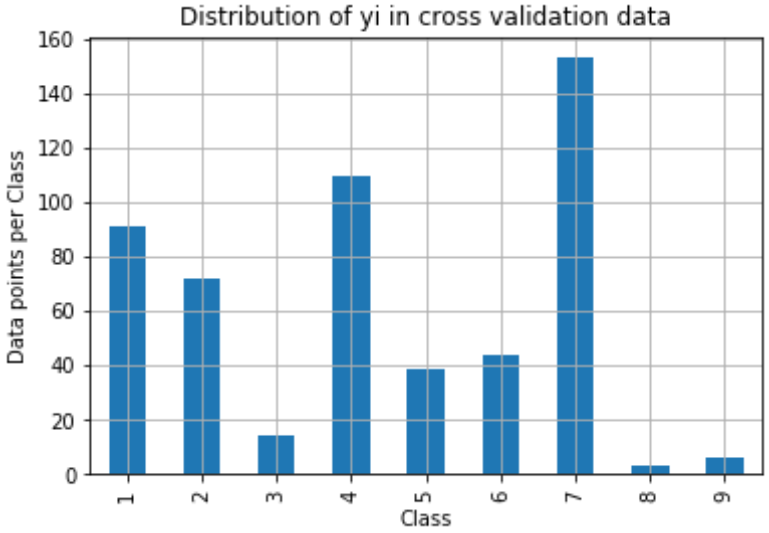



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 [0]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of 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 [0]:

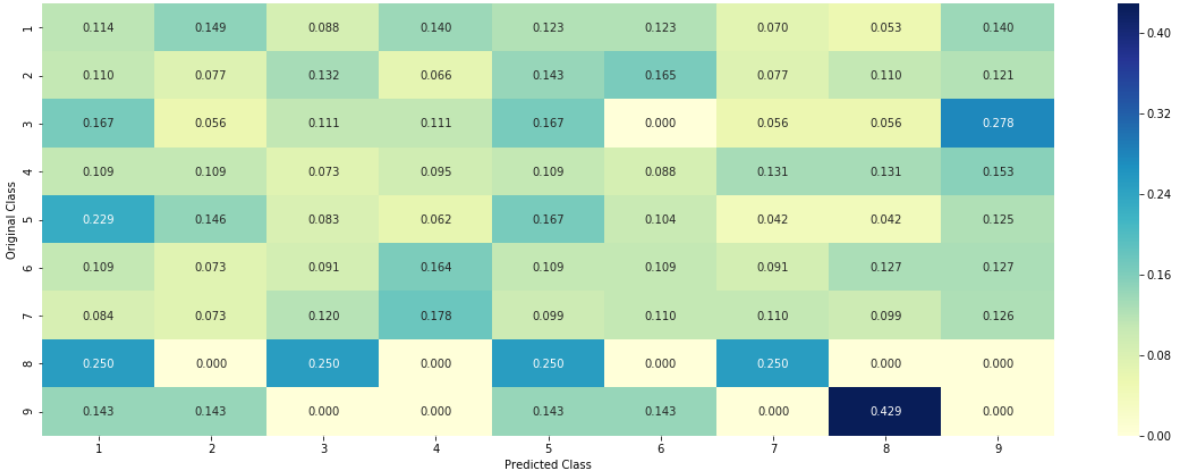
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

```
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_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 2.42076828262288
Log loss on Test Data using Random Model 2.5447412990852283
-------------------- Confusion matrix --------------------
```


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

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



3.3 Univariate Analysis

In [0]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
```

```
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given 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 [0]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
```

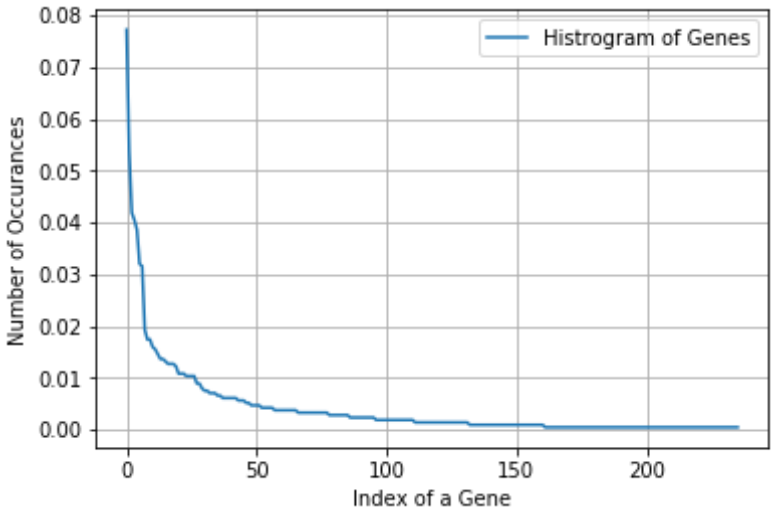
```
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 236
BRCA1     164
TP53      114
BRCA2      89
PTEN       86
EGFR       82
KIT        68
BRAF       67
ALK        41
PIK3CA     37
PDGFRA     37
Name: Gene, dtype: int64
```

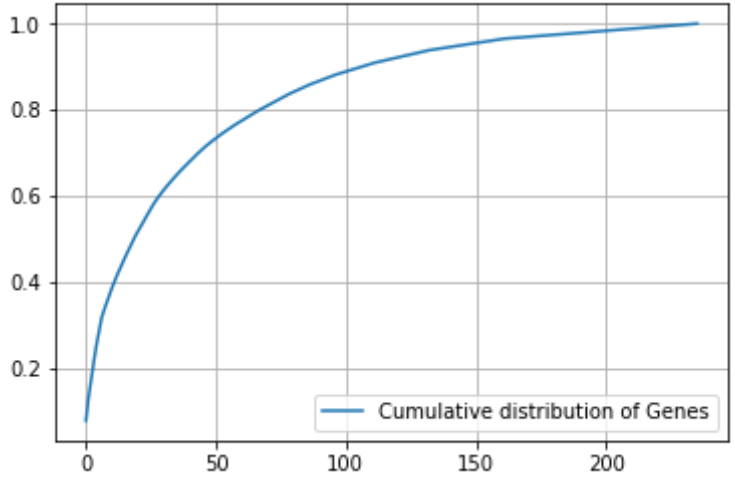
In [0]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

```
Ans: There are 236 different categories of genes in the train data, and
they are distibuted as follows
```

In [0]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [0]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [0]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "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 [0]:
```
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 [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(max_features =1000)
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 [0]:
```
train_df['Gene'].head()
```

Out[0]:
```
2085       AGO2
3078     NOTCH1
679      CDKN2A
2829      BRCA2
301     TMPRSS2
Name: Gene, dtype: object
```

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

Out[0]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
```

```
                'bcl10',
                'bcl2',
                'bcl2l11',
                'bcor',
                'braf',
                'brca1',
                'brca2',
                'brd4',
                'brip1',
                'btk',
                'card11',
                'carm1',
                'casp8',
                'cbl',
                'ccnd1',
                'ccnd3',
                'ccne1',
                'cdh1',
                'cdk12',
                'cdk4',
                'cdk6',
                'cdkn1a',
                'cdkn1b',
                'cdkn2a',
                'cdkn2b',
                'cdkn2c',
                'cebpa',
                'chek2',
                'cic',
                'crebbp',
                'ctcf',
                'ctla4',
                'ctnnb1',
                'ddr2',
                'dicer1',
                'dnmt3a',
                'dnmt3b',
                'egfr',
                'eif1ax',
```

```
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fam58a',
                'fanca',
                'fat1',
                'fbxw7',
                'fgf3',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxp1',
                'fubp1',
                'gata3',
                'gli1',
                'gna11',
                'gnaq',
                'gnas',
                'hist1h1c',
                'hla',
```

```
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikzf1',
                'il7r',
                'inpp4b',
                'jak1',
                'jak2',
                'jun',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
                'klf4',
                'kmt2a',
                'kmt2c',
                'kmt2d',
                'knstrn',
                'kras',
                'lats1',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'mdm4',
                'med12',
                'mef2b',
                'men1',
                'met',
                'mga',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
```

```
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'notch2',
                'npm1',
                'nras',
                'ntrk1',
                'ntrk3',
                'nup93',
                'pak1',
                'pax8',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pik3r3',
                'pim1',
                'pms2',
                'pole',
                'ppm1d',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
```

```
                'rab35',
                'rac1',
                'rad21',
                'rad50',
                'rad51b',
                'rad51c',
                'rad51d',
                'rad54l',
                'raf1',
                'rara',
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rictor',
                'rit1',
                'rnf43',
                'ros1',
                'rras2',
                'runx1',
                'rxra',
                'sdhc',
                'sf3b1',
                'shq1',
                'smad2',
                'smad3',
                'smad4',
                'smarca4',
                'smarcb1',
                'smo',
                'sos1',
                'sox9',
                'spop',
                'srsf2',
                'stat3',
                'stk11',
                'tcf3',
                'tcf7l2',
```

```
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'xpo1',
 'yap1']
```

In [0]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

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

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

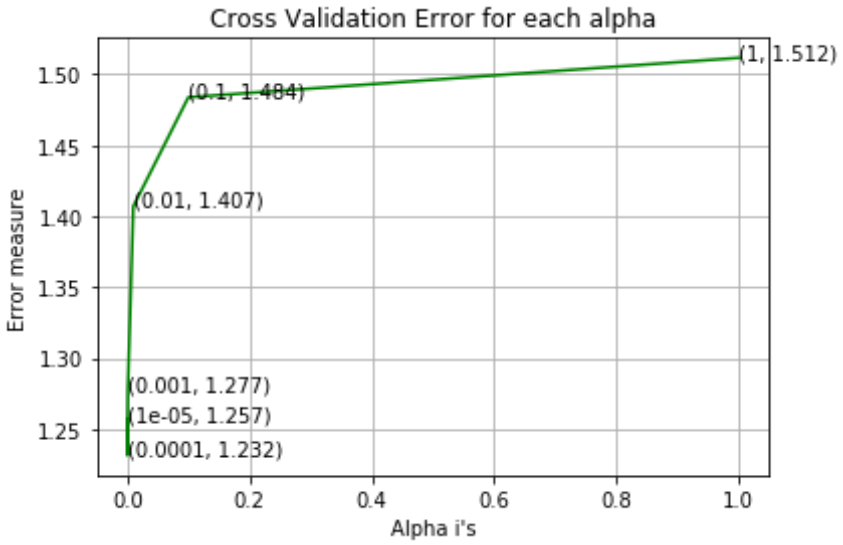
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

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

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

```
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
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: 1.2565685319471083
For values of alpha =  0.0001 The log loss is: 1.2315304035142973
For values of alpha =  0.001 The log loss is: 1.276963618764239
For values of alpha =  0.01 The log loss is: 1.4069344515495001
For values of alpha =  0.1 The log loss is: 1.4837899478090806
For values of alpha =  1 The log loss is: 1.5115978327201767
```




```
For values of best alpha =  0.0001 The train log loss is: 0.977028635
0990097
For values of best alpha =  0.0001 The cross validation log loss is:
1.2315304035142973
For values of best alpha =  0.0001 The test log loss is: 1.2228332498
477632
```

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

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

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

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'
]))))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].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
6  genes in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 515 out of  532 : 96.80451127819549
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

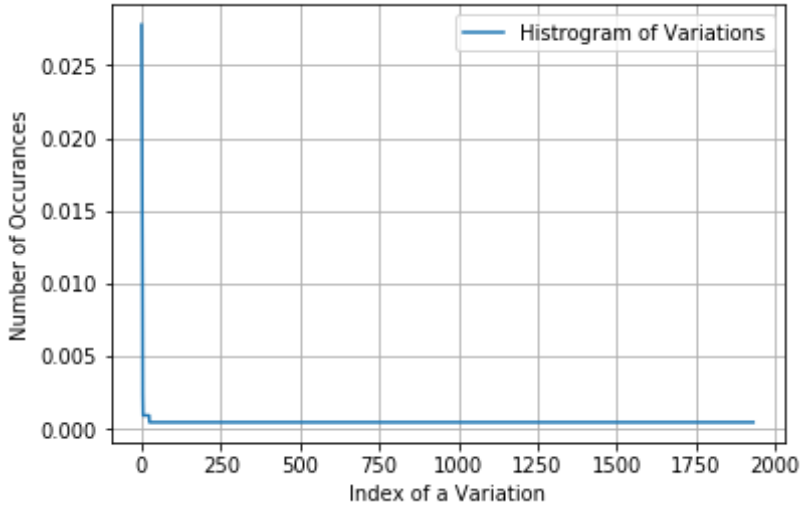
In [0]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1932
Truncating_Mutations    59
Deletion                49
Amplification           42
Fusions                 22
Overexpression           5
F28L                     2
Y64A                     2
Q209L                    2
R170W                    2
Q61K                     2
Name: Variation, dtype: int64
```

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

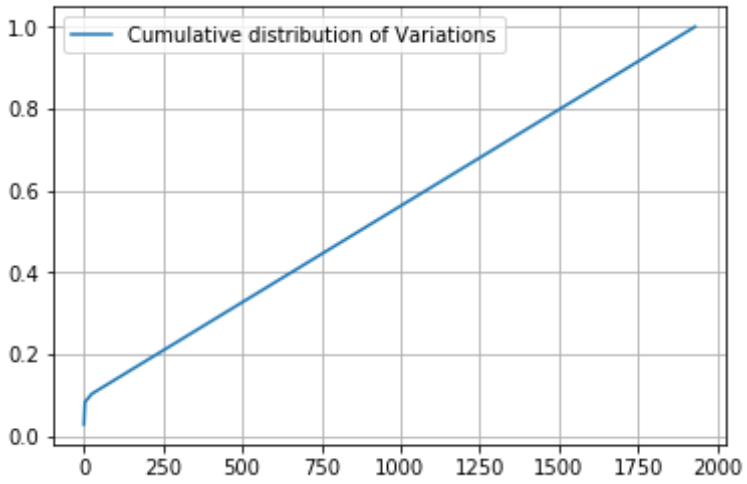
```
Ans: There are 1932 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [0]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [0]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02777778 0.05084746 0.07062147 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [0]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
```

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

In [0]:
```
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 [0]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(max_features=1000)
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 [0]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_onehotCoding.shape)
```

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

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

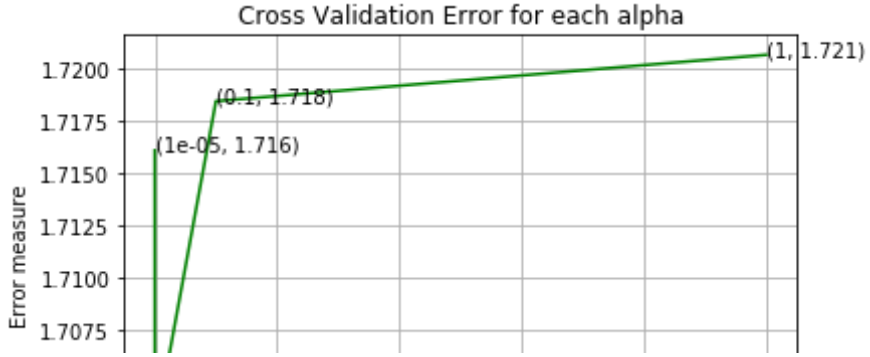
Let's build a model just like the earlier!

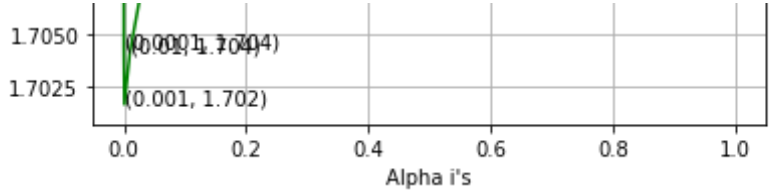
In [0]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/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_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.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_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: 1.7161079361526252
For values of alpha =  0.0001 The log loss is: 1.7043702670186918
For values of alpha =  0.001 The log loss is: 1.7016201632670398
For values of alpha =  0.01 The log loss is: 1.7041255317917239
For values of alpha =  0.1 The log loss is: 1.7184695586177605
For values of alpha =  1 The log loss is: 1.7206921005156055
```






```
For values of best alpha =  0.001 The train log loss is: 1.393244553763

5654
For values of best alpha =  0.001 The cross validation log loss is: 1.7
016201632670398
For values of best alpha =  0.001 The test log loss is: 1.7149210446288
785
```

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

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

In [0]:
```
print("Q12. How many data points are covered by total ", unique_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  1932  genes in test and
cross validation data sets?
Ans
1. In test data 58 out of 665 : 8.721804511278195
2. In cross validation data 69 out of  532 : 12.969924812030076
```

3.2.3 Univariate Analysis on Text Feature

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

In [0]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
```

In [0]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(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 [0]:
```
# building a CountVectorizer with all the words that occured minimum 3
```

```
 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
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 : 1000
```

In [0]:

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

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


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
```

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

In [0]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [0]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [0]:
```
# 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 [0]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
```

```
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

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

```
Counter({250.66566954056066: 1, 175.6178250838014: 1, 134.4953179858607
4: 1, 131.14980595598965: 1, 125.53655859291514: 1, 114.93057757895289:
1, 114.67412930784664: 1, 114.09923393670509: 1, 107.66589717628558: 1,
107.44672585150727: 1, 105.85465729951278: 1, 92.16261892840674: 1, 88.
95856593722557: 1, 87.54526767693956: 1, 86.20869055525652: 1, 80.51727
80726817: 1, 78.68206745153472: 1, 77.43933266781731: 1, 76.83336829621
453: 1, 74.89622525007817: 1, 74.00207664333482: 1, 73.0581906404865:
1, 70.5979732318966: 1, 67.62987393374188: 1, 66.94053731153728: 1, 66.
06876091308463: 1, 65.74890527519419: 1, 65.2049254334314: 1, 63.422415
072818005: 1, 62.716899823288635: 1, 62.24835713100384: 1, 62.162915938
75682: 1, 60.72828066600787: 1, 59.90011225047962: 1, 57.9647898508568
7: 1, 57.82306001655033: 1, 57.589769376392105: 1, 55.817676444240625:
1, 54.75412573374738: 1, 52.959736824975494: 1, 50.57018911651985: 1, 4
8.768231309719916: 1, 48.43069211484466: 1, 48.36707361149832: 1, 48.29
899416145131: 1, 48.066491477480845: 1, 47.63694970602341: 1, 45.272322
97439325: 1, 45.06250670263656: 1, 44.94476344526951: 1, 44.82445249809
456: 1, 43.9617633356787: 1, 43.368368815955655: 1, 42.55139041284737:
1, 42.50209831550633: 1, 42.13562787258803: 1, 42.100750832387625: 1, 4
1.86834676804909: 1, 41.743785889310075: 1, 41.33933332235707: 1, 41.22
98296150011: 1, 41.22651498104832: 1, 41.12796188980745: 1, 41.10760342
8674224: 1, 40.56256381661012: 1, 39.087248913066134: 1, 38.69492406316
418: 1, 38.66890728822808: 1, 38.24313132809187: 1, 38.19229423607635:
1, 38.11034629955275: 1, 37.92136621566518: 1, 37.81767906373868: 1, 3
7.630938783896944: 1, 37.60197354687247: 1, 37.427292687393404: 1, 37.1
05971287213904: 1, 36.29787090550967: 1, 35.89391126038543: 1, 35.70666
246918078: 1, 35.570355677226765: 1, 35.36221680916548: 1, 34.732798911
22236: 1, 34.39895437411477: 1, 34.39115663042189: 1, 34.3331591546920
6: 1, 34.14923583759009: 1, 33.999461934575734: 1, 33.87477364072734:
1, 33.02147307257018: 1, 32.64415147414242: 1, 32.50094390326071: 1, 3
2.25648978724166: 1, 32.196767263917806: 1, 32.16320052337096: 1, 32.15
4756825184236: 1, 32.10805165393714: 1, 31.697040042291746: 1, 31.57571
7246484924: 1, 31.35407114358227: 1, 31.272164738535963: 1, 31.20549544
108645: 1, 31.126626474307443: 1, 31.099589156263768: 1, 31.05797102859
```

591: 1, 31.047413860147724: 1, 30.900298973335282: 1, 30.7062473815600
9: 1, 30.604523270817353: 1, 30.385440461750772: 1, 30.38013064970025:
1, 30.27636208637782: 1, 30.04815516558612: 1, 29.85631497397374: 1, 2
9.85407890184665: 1, 29.74515526539904: 1, 29.737730663916775: 1, 29.60
8148088653603: 1, 29.492472261327926: 1, 29.31437835160415: 1, 29.23223
995661801: 1, 29.16174304779251: 1, 28.779784461742395: 1, 28.605700795
826348: 1, 28.564469888372038: 1, 28.488088130512484: 1, 28.40214850942
981: 1, 28.376572121993686: 1, 28.365009842904666: 1, 27.87337420000871
2: 1, 27.82811061124943: 1, 27.611610781866307: 1, 27.449080539411323:
1, 27.25776962509459: 1, 27.250664159234603: 1, 27.218920213287788: 1,
26.919197458829665: 1, 26.843491697337086: 1, 26.72155842257182: 1, 26.
64990716540107: 1, 26.583687841167396: 1, 26.57716410427115: 1, 26.4476
38834705362: 1, 26.129864482472318: 1, 26.125681739483404: 1, 26.095346
83434214: 1, 26.000920644123894: 1, 25.97243264125008: 1, 25.9498484599
7585: 1, 25.547688666160155: 1, 25.481634306108656: 1, 25.4237919004754
8: 1, 25.289395603258466: 1, 25.288292830234976: 1, 25.226302288435274:
1, 25.180690431247726: 1, 25.024159215712263: 1, 24.936165355107875: 1,
24.815005536901126: 1, 24.728535221438452: 1, 24.713345788768162: 1, 2
4.505627521941967: 1, 24.261889299055312: 1, 24.246441984121702: 1, 24.
141549874331204: 1, 24.07211710449341: 1, 23.963287570220025: 1, 23.918
841095552104: 1, 23.843466219158632: 1, 23.718171165686172: 1, 23.67535
99054071: 1, 23.651578285920024: 1, 23.49841636044465: 1, 23.4732138137
05563: 1, 23.439200316529845: 1, 23.409725215784725: 1, 23.381456707319
24: 1, 23.37411810093746: 1, 23.317056035791026: 1, 23.304935920441995:
1, 23.178647238224173: 1, 23.025157205557516: 1, 23.00247379217369: 1,
22.98191210400342: 1, 22.90759696945325: 1, 22.893710049651197: 1, 22.6
90978320809304: 1, 22.656201238045: 1, 22.56921462271021: 1, 22.5627075
1904388: 1, 22.44096882449039: 1, 22.403251495567023: 1, 22.32473025710
977: 1, 22.235129839976494: 1, 22.195796532753207: 1, 22.17728718472922
7: 1, 22.152442572698487: 1, 22.144228417494627: 1, 22.048083829336527:
1, 21.91294006943531: 1, 21.874320779340337: 1, 21.81772080156392: 1, 2
1.780519871963453: 1, 21.725644728285143: 1, 21.640397666238357: 1, 21.
628908479593594: 1, 21.617593312027335: 1, 21.59575247974748: 1, 21.548
75261781468: 1, 21.536976703088882: 1, 21.433102878651862: 1, 21.418991
724954076: 1, 21.162412339374654: 1, 21.145796966298207: 1, 21.12326795
858357: 1, 21.10494733996432: 1, 21.103146688559807: 1, 21.091597979407
712: 1, 21.07051479601072: 1, 21.058148485524676: 1, 20.98723878953415:
1, 20.977790215955334: 1, 20.959195066178733: 1, 20.937683562093408: 1,
20.909216559672572: 1, 20.884748123916335: 1, 20.87724866205609: 1, 20.

```
86521401072492: 1, 20.828342916192728: 1, 20.823597955006928: 1, 20.740
19187442016: 1, 20.669245614837227: 1, 20.627681113705272: 1, 20.599281
988355536: 1, 20.57984562081354: 1, 20.568551415894046: 1, 20.495508249
591776: 1, 20.384176963410454: 1, 20.241000982963573: 1, 20.23480059094
717: 1, 20.184987332041427: 1, 20.18142757973565: 1, 20.17140487608213
6: 1, 20.14126892410702: 1, 20.062219477180726: 1, 20.05479335005742:
1, 20.047112437776853: 1, 20.01880577400343: 1, 19.97976954157108: 1, 1
9.777236670205173: 1, 19.708508170226736: 1, 19.64187964262621: 1, 19.6
27390643937346: 1, 19.612993353848722: 1, 19.58948502592861: 1, 19.4875
9406324506: 1, 19.47717166712504: 1, 19.469232779540448: 1, 19.45404380
098142: 1, 19.3933440100751: 1, 19.284740010981427: 1, 19.2054933602807
98: 1, 19.196418812612325: 1, 19.13268207522189: 1, 19.116798046279794:
1, 19.069147504958636: 1, 19.04664084670654: 1, 19.0427364440518: 1, 1
9.013253049907583: 1, 19.01079726722704: 1, 18.962524232933642: 1, 18.9
18554841847868: 1, 18.891300328144457: 1, 18.871911883255542: 1, 18.858
36257444061: 1, 18.85065396055366: 1, 18.804775857556304: 1, 18.7426263
3337917: 1, 18.725651304118777: 1, 18.68462649394145: 1, 18.67925731043
646: 1, 18.678432438883217: 1, 18.644785707189737: 1, 18.51618582673903
3: 1, 18.498642731150213: 1, 18.483508900227243: 1, 18.472846255575494:
1, 18.43570626555932: 1, 18.43154172709025: 1, 18.37359208353564: 1, 1
8.280941469785077: 1, 18.264072101813028: 1, 18.244138784671552: 1, 18.
228882393141237: 1, 18.164037708959913: 1, 18.140879083565085: 1, 18.13
920135041355: 1, 18.13691768591495: 1, 18.07987413884237: 1, 18.0310022
10334663: 1, 17.902601962440826: 1, 17.898861260829054: 1, 17.854114567
4115: 1, 17.842728812486556: 1, 17.80854351396389: 1, 17.78577194277529
5: 1, 17.78459510788749: 1, 17.772446800169547: 1, 17.737461031289993:
1, 17.670264694374335: 1, 17.628139836645538: 1, 17.62552162578866: 1,
17.53717700780554: 1, 17.508817548272336: 1, 17.402883389280674: 1, 17.
389986831102593: 1, 17.35656849260737: 1, 17.344523058404498: 1, 17.300
038469262358: 1, 17.256491591202085: 1, 17.25243753599014: 1, 17.218327
69635131: 1, 17.209246786700426: 1, 17.15830540431092: 1, 17.1527808040
66495: 1, 17.151285708353132: 1, 17.140449330062214: 1, 17.135927732368
383: 1, 17.11645441112545: 1, 17.099672132987234: 1, 17.00779575601985
3: 1, 17.001161303076746: 1, 16.981709291274314: 1, 16.941634621977098:
1, 16.889589902320555: 1, 16.887665450522245: 1, 16.879085349374215: 1,
16.875961343858318: 1, 16.7899334131068: 1, 16.752821766946045: 1, 16.7
087358313194: 1, 16.705310360459073: 1, 16.69368082343257: 1, 16.692961
96283079: 1, 16.644205264893237: 1, 16.637168557414668: 1, 16.550957076
383625: 1, 16.50479499513197: 1, 16.47205491938089: 1, 16.4705486573605
```

```
4: 1, 16.45243452041546: 1, 16.407281441517892: 1, 16.380489111043165:
1, 16.368145677774116: 1, 16.36811064802907: 1, 16.367996752230965: 1,
16.346294181954413: 1, 16.293811782281175: 1, 16.291419520709024: 1, 1
6.2901755213852: 1, 16.27247627685998: 1, 16.252330266709556: 1, 16.163
970718216547: 1, 16.161907534772528: 1, 16.14282847304037: 1, 16.119694
490724033: 1, 16.114760440245657: 1, 16.110791499125504: 1, 16.10920195
7724053: 1, 16.08185575913704: 1, 16.04721475541772: 1, 16.021725256467
207: 1, 15.987345605844135: 1, 15.983870195545085: 1, 15.98098231179612
6: 1, 15.90669158846064: 1, 15.888425223308152: 1, 15.86896358177527:
1, 15.84153950311088: 1, 15.828796748541862: 1, 15.77531904294378: 1, 1
5.745088718204906: 1, 15.725937828126716: 1, 15.667065900102005: 1, 15.
663139367105725: 1, 15.580464238893628: 1, 15.533293194143663: 1, 15.49
8823134997801: 1, 15.487717822503816: 1, 15.456290006607716: 1, 15.4485
34505130283: 1, 15.377336366581067: 1, 15.373084491509251: 1, 15.372693
746542941: 1, 15.341376070008577: 1, 15.333754857571382: 1, 15.28734352
7456107: 1, 15.253472621381329: 1, 15.207925839254932: 1, 15.1959564426
18401: 1, 15.180651185000578: 1, 15.176378329274666: 1, 15.119098676459
64: 1, 15.112101973357692: 1, 15.012993549494894: 1, 14.97858025878206
3: 1, 14.974838765762833: 1, 14.952452708290531: 1, 14.944106135255426:
1, 14.937812544370827: 1, 14.919831610405152: 1, 14.883062434504403: 1,
14.875880754999404: 1, 14.804248914974899: 1, 14.782049704880219: 1, 1
4.773246167027: 1, 14.76316846358955: 1, 14.747618423400871: 1, 14.6933
4072822519: 1, 14.69287724808427: 1, 14.668101586497201: 1, 14.64271535
5574575: 1, 14.617793710693403: 1, 14.608376388534358: 1, 14.6025634365
40152: 1, 14.595861709898314: 1, 14.568541470463735: 1, 14.559500519041
82: 1, 14.547304150152472: 1, 14.534995545871329: 1, 14.49822115133676
8: 1, 14.464980483754315: 1, 14.450497264085405: 1, 14.44642942538349:
1, 14.441843024861338: 1, 14.436379064041136: 1, 14.419034548073217: 1,
14.418157798367115: 1, 14.41324365032414: 1, 14.378924445017468: 1, 14.
368330436468202: 1, 14.334131678030559: 1, 14.33261794136867: 1, 14.302
030768579609: 1, 14.279963691486227: 1, 14.267118445455969: 1, 14.25933
357860938: 1, 14.258673269978534: 1, 14.25856038138067: 1, 14.235658711
938477: 1, 14.206890797793704: 1, 14.20611623737805: 1, 14.197055300426
571: 1, 14.17611780917039: 1, 14.169325391044138: 1, 14.14630785347638
7: 1, 14.144188520052513: 1, 14.133401917518642: 1, 14.131582604858629:
1, 14.125841575846032: 1, 14.119041117921766: 1, 14.012754310412666: 1,
13.963917944047099: 1, 13.92226099158804: 1, 13.895519634663508: 1, 13.
868057989911557: 1, 13.850413004753264: 1, 13.847175018558346: 1, 13.84
5622769125914: 1, 13.840811777320882: 1, 13.801691119242733: 1, 13.7361
```

```
49111212645: 1, 13.735582119852468: 1, 13.723364085802368: 1, 13.717800
93571232: 1, 13.694752505038272: 1, 13.691545374343796: 1, 13.661860358
704251: 1, 13.651009159603216: 1, 13.638157603819707: 1, 13.61228139008
8755: 1, 13.58761591952228: 1, 13.580350505109983: 1, 13.56302079081749
7: 1, 13.542619790135301: 1, 13.515278164881888: 1, 13.47008474779349:
1, 13.419890300481873: 1, 13.393079601951616: 1, 13.391576783040165: 1,
13.351690708244922: 1, 13.331902555223822: 1, 13.298706428288948: 1, 1
3.2798373307685: 1, 13.26781390639167: 1, 13.237690560334174: 1, 13.178
864983127184: 1, 13.15844699106357: 1, 13.158290710601268: 1, 13.142137
082236813: 1, 13.096380962315067: 1, 13.088627328366305: 1, 13.04357232
9718952: 1, 13.027569710235362: 1, 13.01266034772092: 1, 12.98261371424
777: 1, 12.957563656781723: 1, 12.941025826048017: 1, 12.92675102428343
9: 1, 12.923898735313376: 1, 12.917196094548734: 1, 12.875490415228029:
1, 12.85327489130538: 1, 12.850641530946747: 1, 12.77910159094578: 1, 1
2.759818693904592: 1, 12.75304602744065: 1, 12.752113318337795: 1, 12.6
94549482593946: 1, 12.6893069998733: 1, 12.675609337200102: 1, 12.66731
201993778: 1, 12.666176033967279: 1, 12.661072785397991: 1, 12.65486415
6851373: 1, 12.594564541658338: 1, 12.582157237375885: 1, 12.5096355198
9031: 1, 12.504464047934432: 1, 12.45593587820471: 1, 12.44893119575496
4: 1, 12.431220513199815: 1, 12.385415526918871: 1, 12.383043694634663:
1, 12.376029826048182: 1, 12.363902938011424: 1, 12.348574278458146: 1,
12.348306386443584: 1, 12.34462911424279: 1, 12.31795003475475: 1, 12.2
88958292877012: 1, 12.268874457763141: 1, 12.256095788361304: 1, 12.184
257792422263: 1, 12.179304507378294: 1, 12.174316827670246: 1, 12.13152
6369460968: 1, 12.109936839659632: 1, 12.071024417680317: 1, 12.0128162
74457984: 1, 11.991269784204734: 1, 11.975566865408567: 1, 11.967398269
006475: 1, 11.944403249560366: 1, 11.932023186754725: 1, 11.90720537750
2091: 1, 11.888107891057476: 1, 11.874388456881277: 1, 11.8376508798854
09: 1, 11.835409691306932: 1, 11.823759547906041: 1, 11.82332749490429
4: 1, 11.81479482791322: 1, 11.81129261019041: 1, 11.790856530056946:
1, 11.790170957209265: 1, 11.775363821891723: 1, 11.772100936476775: 1,
11.763858147334858: 1, 11.735649696785995: 1, 11.720543629085777: 1, 1
1.71599451118974: 1, 11.714213001544458: 1, 11.688859597846248: 1, 11.6
82635387716246: 1, 11.67436235523935: 1, 11.650681593536445: 1, 11.6326
24497423704: 1, 11.59941605839937: 1, 11.597527830648325: 1, 11.5954472
52751136: 1, 11.585776112257685: 1, 11.584160762496706: 1, 11.561958354
667182: 1, 11.495172289261482: 1, 11.477434684857194: 1, 11.46414560505
4544: 1, 11.449680865521831: 1, 11.446501319694685: 1, 11.4420041350161
95: 1, 11.433747250912086: 1, 11.42510554881092: 1, 11.423621196574159:
```

1, 11.412012299315055: 1, 11.405032539686301: 1, 11.390774331040236: 1,
11.390542103179083: 1, 11.376421887622636: 1, 11.376073455188052: 1, 1
1.330670614809815: 1, 11.286920561795233: 1, 11.22837518146306: 1, 11.2
2744739555307: 1, 11.22359093370738: 1, 11.221728334118344: 1, 11.21429
4414036315: 1, 11.206932342822345: 1, 11.189524378943412: 1, 11.1676913
86306876: 1, 11.147880039361427: 1, 11.124617779959582: 1, 11.124350258
386253: 1, 11.108534156522321: 1, 11.107726417653138: 1, 11.08119560120
5199: 1, 11.06405850230914: 1, 11.028539594782877: 1, 11.01271898419090
6: 1, 10.989041487157353: 1, 10.978312679232522: 1, 10.94653112954738:
1, 10.93073735699152: 1, 10.926406754313195: 1, 10.914355523884279: 1,
10.895317064708175: 1, 10.888488497655267: 1, 10.878407400347541: 1, 1
0.8783393825914: 1, 10.877953356842704: 1, 10.873592433239857: 1, 10.86
219211760146: 1, 10.857934467567729: 1, 10.855403567521268: 1, 10.82200
1338606595: 1, 10.817114853583552: 1, 10.794855629098189: 1, 10.7879573
08066845: 1, 10.778182392296944: 1, 10.76665965525976: 1, 10.7557126641
31298: 1, 10.745595615174176: 1, 10.736337836925301: 1, 10.731635948422
499: 1, 10.72375807785566: 1, 10.718143625925201: 1, 10.71097429549006:
1, 10.704327167104069: 1, 10.701273376089723: 1, 10.698398137078732: 1,
10.697119278311215: 1, 10.68921401472196: 1, 10.673462637139576: 1, 10.
671829360528264: 1, 10.641832597898839: 1, 10.640108792391825: 1, 10.64
0072865949564: 1, 10.63649790535517: 1, 10.615386026974903: 1, 10.61536
1922114943: 1, 10.61405459399028: 1, 10.607805405854718: 1, 10.60598214
234185: 1, 10.556140302717699: 1, 10.555063088019704: 1, 10.54861358768
5082: 1, 10.54639408181235: 1, 10.526042206123543: 1, 10.52336146807617
8: 1, 10.52224416602508: 1, 10.519764043828827: 1, 10.505973829975483:
1, 10.496574370328581: 1, 10.484642727815151: 1, 10.473427369293459: 1,
10.472448297470235: 1, 10.464911089056534: 1, 10.453583957958314: 1, 1
0.430223154272184: 1, 10.429128537026543: 1, 10.419164339110411: 1, 10.
40950408869963: 1, 10.40523524710077: 1, 10.384797306840415: 1, 10.3456
77507596985: 1, 10.340372940456179: 1, 10.336575093999713: 1, 10.312935
805939674: 1, 10.308972950293132: 1, 10.286862079278414: 1, 10.28231557
8085061: 1, 10.27983120558689: 1, 10.24951677300139: 1, 10.201282729066
504: 1, 10.193316367264512: 1, 10.166289151154405: 1, 10.1462549752882
6: 1, 10.135031082885678: 1, 10.119774224612314: 1, 10.105355755092631:
1, 10.07205956398631: 1, 10.067324919965515: 1, 10.044920260385686: 1,
10.036391612599438: 1, 10.027144368188775: 1, 10.013671193945596: 1, 9.
998982594869508: 1, 9.979859081733157: 1, 9.976948071868335: 1, 9.95866
4781751562: 1, 9.954845096997165: 1, 9.933632811905337: 1, 9.9324482617
68297: 1, 9.929027447803202: 1, 9.926336127541424: 1, 9.90314545638160

7: 1, 9.898604518037855: 1, 9.883551856022287: 1, 9.873469510697067: 1,
9.864513016904963: 1, 9.840864882046363: 1, 9.830456664050207: 1, 9.816
950259929534: 1, 9.813962669317887: 1, 9.813457517327082: 1, 9.80842604
868881: 1, 9.80649043279285: 1, 9.805293792491993: 1, 9.80149855321195
9: 1, 9.793458023039078: 1, 9.786516464258508: 1, 9.78293808256314: 1,
9.778022606815567: 1, 9.772797850368919: 1, 9.749031556860826: 1, 9.746
458692299386: 1, 9.745006939398792: 1, 9.739188301772124: 1, 9.72343946
2322535: 1, 9.7213763056776: 1, 9.71146744288984: 1, 9.707961451409636:
1, 9.698046126471802: 1, 9.690222861226447: 1, 9.683312057230921: 1, 9.
683250288298067: 1, 9.674826943681238: 1, 9.631412696658275: 1, 9.63130
1548063025: 1, 9.619972789208948: 1, 9.602174566439292: 1, 9.5415591429
92991: 1, 9.528656804019715: 1, 9.503865129744275: 1, 9.50198773841475
8: 1, 9.49965006795173: 1, 9.49612104050093: 1, 9.479951049950325: 1,
9.467775005884713: 1, 9.461990345749614: 1, 9.446919116799531: 1, 9.445
082214360006: 1, 9.412018967120902: 1, 9.401973707400494: 1, 9.39500197
090001: 1, 9.390698005074603: 1, 9.38845745324861: 1, 9.38719646241879:
1, 9.382546509362815: 1, 9.358012285579106: 1, 9.351185099640453: 1, 9.
3475518667094: 1, 9.331328258617495: 1, 9.329066025648794: 1, 9.2999456
40204717: 1, 9.29349919150019: 1, 9.291942108891682: 1, 9.250372978831:
1, 9.24741204775064: 1, 9.242177287021939: 1, 9.233571748097773: 1, 9.2
13264134344955: 1, 9.211009478397065: 1, 9.207800133480891: 1, 9.191043
108942434: 1, 9.166878610825277: 1, 9.13260569092953: 1, 9.120552842841
121: 1, 9.106419079568154: 1, 9.101455516503085: 1, 9.078676205228335:
1, 9.074834320379093: 1, 9.06393393926755: 1, 9.063690708094871: 1, 9.0
5577374036874: 1, 9.047052966287971: 1, 9.042549259222175: 1, 9.0424753
98450506: 1, 9.040305865260596: 1, 9.018806084034841: 1, 8.984917672900
41: 1, 8.97989682591384: 1, 8.97936267734772: 1, 8.96604159791676: 1,
8.959957045569217: 1, 8.957340088060315: 1, 8.951731666970751: 1, 8.950
621786700452: 1, 8.944579927270764: 1, 8.941056670467965: 1, 8.92295638
3981116: 1, 8.92284995466804: 1, 8.922648876247322: 1, 8.91975552817465
8: 1, 8.91455473858006: 1, 8.902959263745993: 1, 8.902412280469669: 1,
8.895793583126903: 1, 8.890615077069288: 1, 8.88722571947839: 1, 8.8769
62934423863: 1, 8.872378948319337: 1, 8.866898004995788: 1, 8.857171489
374512: 1, 8.8466916993766: 1, 8.844048894934371: 1, 8.837742638992085:
1, 8.834118873569963: 1, 8.796685742084165: 1, 8.795742259515164: 1, 8.
795139828750704: 1, 8.792230574817381: 1, 8.785782189991448: 1, 8.77387
6120661074: 1, 8.762016372543995: 1, 8.752749340917564: 1, 8.7306298392
03312: 1, 8.717728016917247: 1, 8.714690227826727: 1, 8.71344360919982
6: 1, 8.713244364639996: 1, 8.709732572905873: 1, 8.702008128197011: 1,

8.682725243520562: 1, 8.670843258233191: 1, 8.66700723887834: 1, 8.6650
7312161: 1, 8.658916042636323: 1, 8.65620390998228: 1, 8.63051985668258
2: 1, 8.625143225524173: 1, 8.61971045539538: 1, 8.613302781177499: 1,
8.595692603396593: 1, 8.591311692768802: 1, 8.59095137126867: 1, 8.5663
11167602747: 1, 8.543817130814695: 1, 8.530528803329862: 1, 8.530496091
474678: 1, 8.495641246673504: 1, 8.470241698516544: 1, 8.46919434324949
8: 1, 8.460269700363336: 1, 8.45730102637593: 1, 8.442806802667523: 1,
8.442714639074287: 1, 8.424381339667388: 1, 8.41975733966146: 1, 8.3925
82746979658: 1, 8.387949714993852: 1, 8.37892584910507: 1, 8.3577953916
84778: 1, 8.349746317524584: 1, 8.335933000423594: 1, 8.31547773055114
3: 1, 8.300236244931279: 1, 8.277852267090148: 1, 8.275957074622479: 1,
8.265639554801492: 1, 8.248896868425378: 1, 8.239027341182494: 1, 8.224
114601795227: 1, 8.22138394197732: 1, 8.207820427658444: 1, 8.192998710
865687: 1, 8.187563819878722: 1, 8.183565038574804: 1, 8.17564680179297
4: 1, 8.173782173841476: 1, 8.138089290892507: 1, 8.12618296305547: 1,
8.11550685070518: 1, 8.10918900028868: 1, 8.101201716969884: 1, 8.09842
674948822: 1, 8.096345611916943: 1, 8.094640369193256: 1, 8.09366936357
3577: 1, 8.091183290760197: 1, 8.088681078306054: 1, 8.073356316592639:
1, 8.060759541133951: 1, 8.059310864969014: 1, 8.058677176073047: 1, 8.
046065442610063: 1, 8.03863318291232: 1, 8.03730155853512: 1, 8.0246271
18570068: 1, 8.019190253387773: 1, 7.981501309606565: 1, 7.975048119591
4436: 1, 7.9721911096550935: 1, 7.966408039083885: 1, 7.92710476550849
1: 1, 7.873842896158985: 1, 7.843110376657701: 1, 7.835631391417111: 1,
7.832972663234989: 1, 7.831726335020607: 1, 7.829169684361168: 1, 7.823
615690939895: 1, 7.816934897722398: 1, 7.811000940647976: 1, 7.80095020
427667: 1, 7.791611392033571: 1, 7.790815951755678: 1, 7.77625238151932
7: 1, 7.7735374489860956: 1, 7.743282506474582: 1, 7.741855859698812:
1, 7.74032983779807: 1, 7.730563109482859: 1, 7.72150720603187: 1, 7.71
8334343166488: 1, 7.7176309293504355: 1, 7.710983312832815: 1, 7.691154
376634514: 1, 7.67229207448477: 1, 7.646557578706821: 1, 7.639350873512
721: 1, 7.634320036474747: 1, 7.616427403934945: 1, 7.611555916997435:
1, 7.593526645695531: 1, 7.582836522348879: 1, 7.563012173470711: 1, 7.
548062022984793: 1, 7.512437133024156: 1, 7.505678349284682: 1, 7.49960
9430986953: 1, 7.494494787380554: 1, 7.4908802485729: 1, 7.473873232219
63: 1, 7.468345835415862: 1, 7.450093265776176: 1, 7.423133575971339:
1, 7.41501230415749: 1, 7.390424415965061: 1, 7.338544005211818: 1, 7.3
33882271150924: 1, 7.326870261741542: 1, 7.309752545514681: 1, 7.300440
1871230105: 1, 7.287923131621619: 1, 7.286674026225969: 1, 7.2790631653
21215: 1, 7.262349414628956: 1, 7.251605293901709: 1, 7.22547255716598

```
7: 1, 7.184602804898371: 1, 7.147607796001188: 1, 7.130967814171262: 1,
7.121593382911922: 1, 7.064458254303401: 1, 7.059915224478303: 1, 7.046
884802777183: 1, 7.036883331058027: 1, 7.015438244482524: 1, 7.00955577
1995563: 1, 7.003059678831033: 1, 6.9743714957537595: 1, 6.956214529015
756: 1, 6.926476425622913: 1, 6.907029596041751: 1, 6.897830912148306:
1, 6.879219964334147: 1, 6.842075700964187: 1, 6.787518674905403: 1, 6.
77908992992304: 1, 6.733058707297722: 1, 6.70582041170854: 1, 6.6621399
1909708: 1, 6.651542510248436: 1, 6.647839961198719: 1, 6.6198169836280
55: 1, 6.6151336286915505: 1, 6.54420521305735: 1, 6.4374092800830045:
1, 6.403922414563225: 1, 5.901595135443461: 1})
```

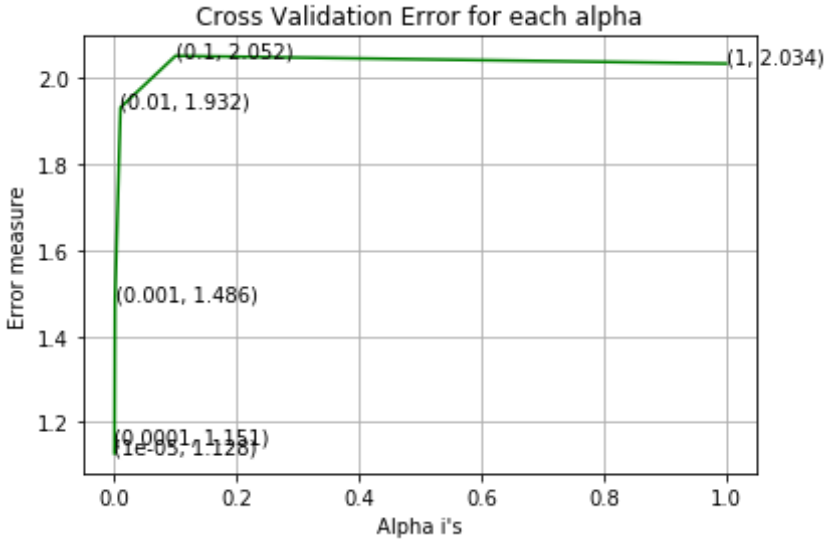
In [0]:

```
# 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: 1.12784680059659
For values of alpha =  0.0001 The log loss is: 1.1510772887943344
For values of alpha =  0.001 The log loss is: 1.485943276123811
For values of alpha =  0.01 The log loss is: 1.9317466588805081


For values of alpha =  0.1 The log loss is: 2.0518401477506174
For values of alpha =  1 The log loss is: 2.0338202985834766
```




```
For values of best alpha =  1e-05 The train log loss is: 0.706096851925
9965
For values of best alpha =  1e-05 The cross validation log loss is: 1.1
2784680059659
For values of best alpha =  1e-05 The test log loss is: 1.1723984905274
08
```

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

Ans. Yes, it seems like!

In [0]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

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

```
94.7 % of word of test data appeared in train data
93.8 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [0]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y,
clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
```

```
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of 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 [0]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [0]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
 or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer(max_features=1000)
    var_count_vec = TfidfVectorizer(max_features=1000)
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
```

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

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

Stacking the three types of features

In [0]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
```

```
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_
feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

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

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


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_x_responseCoding = np.hstack((train_gene_var_responseCoding, trai
n_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_t
ext_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [0]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", 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, 22
35)
(number of data points * number of features) in test data =  (665, 223
5)
(number of data points * number of features) in cross validation data =
(532, 2235)
```

In [0]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", 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, 2
7)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data =
(532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

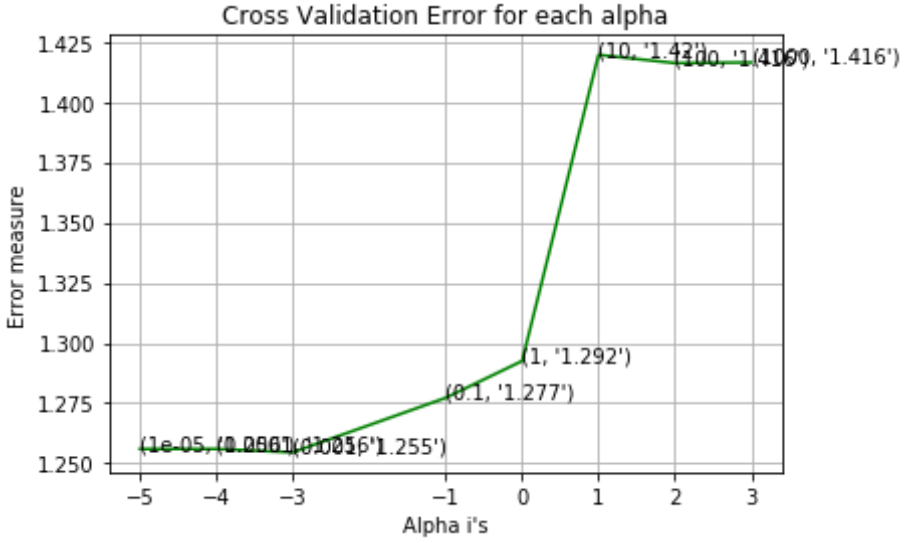
In [0]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
```

```
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 : 1.2559969207147805
for alpha = 0.0001
Log Loss : 1.2559944889725208
for alpha = 0.001
Log Loss : 1.254597097628711
for alpha = 0.1
Log Loss : 1.277193888633039
for alpha = 1
Log Loss : 1.2924835157273473
for alpha = 10
Log Loss : 1.419553471989432
for alpha = 100
Log Loss : 1.416125115229416
for alpha = 1000
Log Loss : 1.4164011415491822
```




```
For values of best alpha =  0.001 The train log loss is: 0.739496360478
5211
For values of best alpha =  0.001 The cross validation log loss is: 1.2
54597097628711
```

```
3459709702871 1
For values of best alpha =  0.001 The test log loss is: 1.2373842541482
603
```

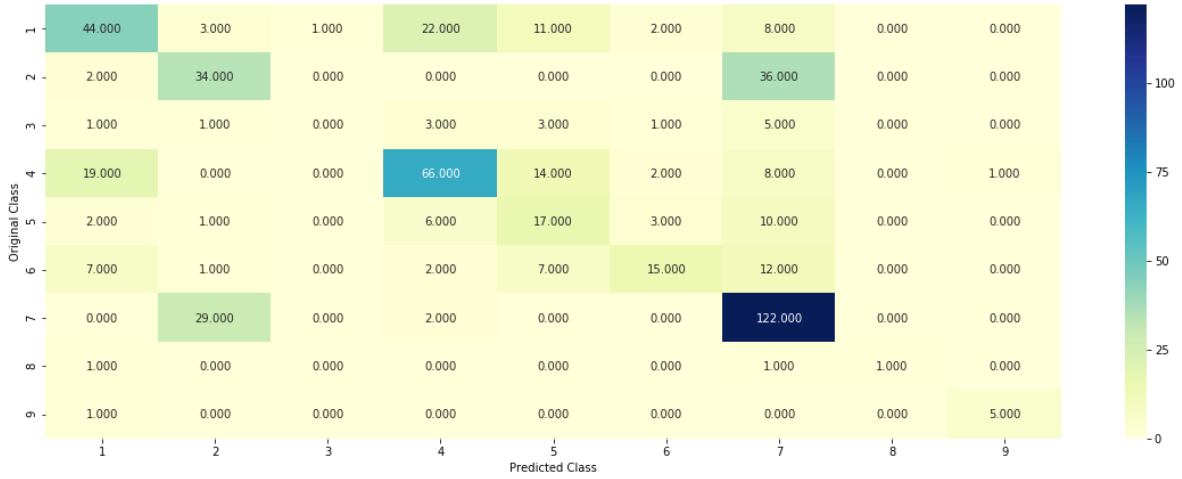
4.1.1.2. Testing the model with best hyper paramters

In [0]:

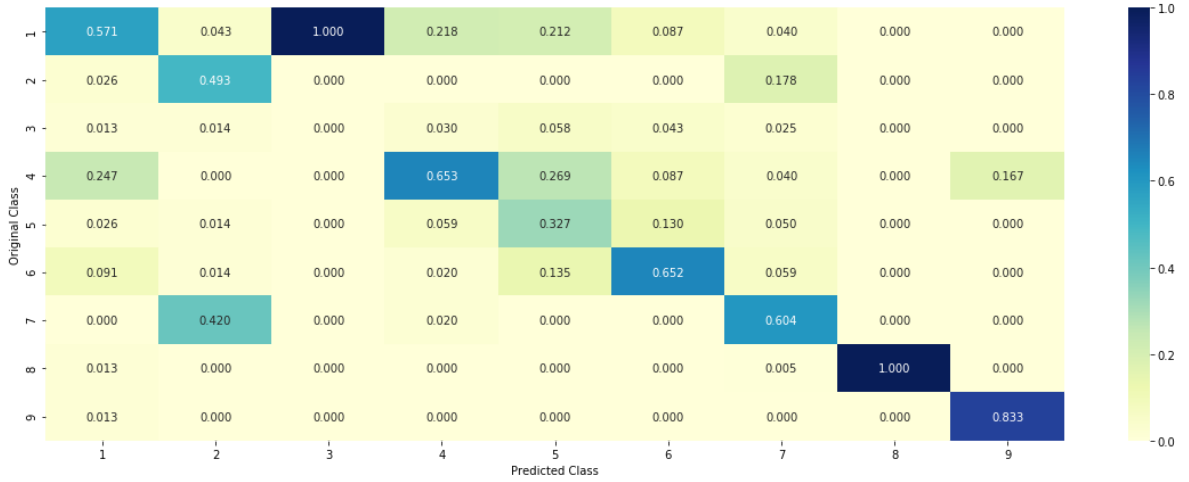
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

# 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
```

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

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-pro
bability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))
```

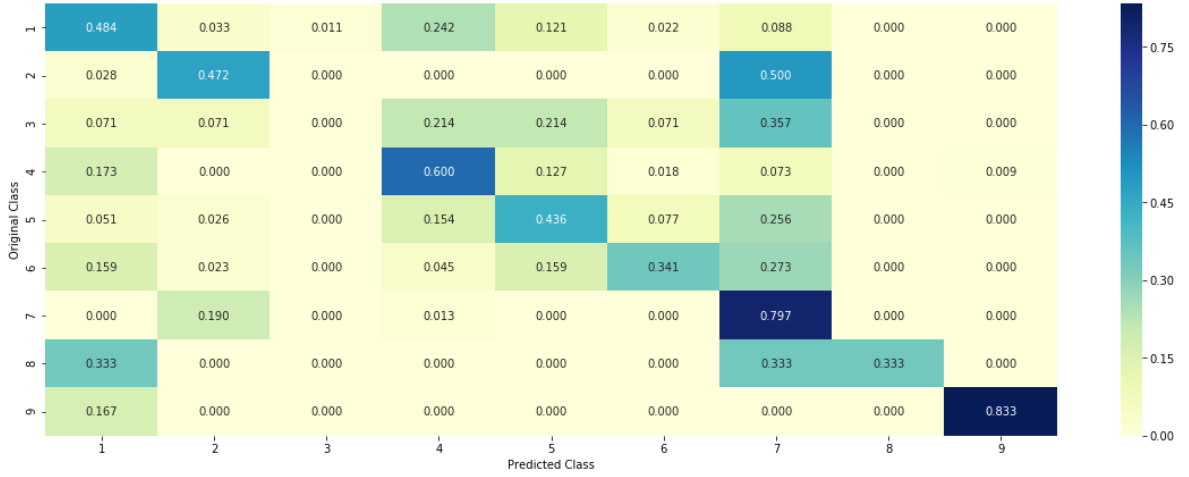
```
Log Loss : 1.254597097628711
Number of missclassified point : 0.42857142857142855
-------------------- Confusion matrix --------------------
```




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



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



4.1.1.3. Feature Importance, Correctly classified point

In [0]:
```
test_point_index = 1
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.057  0.0448 0.0133 0.7461 0.031  0.0
313 0.0698 0.0031 0.0037]]
Actual Class : 5
--------------------------------------------------
11 Text feature [activity] present in test data point [True]
13 Text feature [protein] present in test data point [True]
14 Text feature [pten] present in test data point [True]
16 Text feature [function] present in test data point [True]
17 Text feature [experiments] present in test data point [True]
19 Text feature [results] present in test data point [True]
20 Text feature [acid] present in test data point [True]
21 Text feature [suppressor] present in test data point [True]
22 Text feature [shown] present in test data point [True]
23 Text feature [also] present in test data point [True]
25 Text feature [loss] present in test data point [True]
26 Text feature [type] present in test data point [True]
27 Text feature [amino] present in test data point [True]
28 Text feature [mutations] present in test data point [True]
29 Text feature [missense] present in test data point [True]
30 Text feature [important] present in test data point [True]
32 Text feature [described] present in test data point [True]
33 Text feature [reduced] present in test data point [True]
34 Text feature [wild] present in test data point [True]
35 Text feature [may] present in test data point [True]
37 Text feature [although] present in test data point [True]
38 Text feature [two] present in test data point [True]
40 Text feature [related] present in test data point [True]
```

```
41 Text feature [functional] present in test data point [True]
43 Text feature [determine] present in test data point [True]
45 Text feature [therefore] present in test data point [True]
47 Text feature [30] present in test data point [True]
48 Text feature [indicated] present in test data point [True]
49 Text feature [lower] present in test data point [True]
51 Text feature [previously] present in test data point [True]
53 Text feature [suggesting] present in test data point [True]
54 Text feature [analysis] present in test data point [True]
55 Text feature [either] present in test data point [True]
56 Text feature [thus] present in test data point [True]
58 Text feature [similar] present in test data point [True]
61 Text feature [associated] present in test data point [True]
63 Text feature [phosphatase] present in test data point [True]
65 Text feature [however] present in test data point [True]
66 Text feature [vitro] present in test data point [True]
67 Text feature [figure] present in test data point [True]
69 Text feature [cells] present in test data point [True]
70 Text feature [discussion] present in test data point [True]
74 Text feature [using] present in test data point [True]
75 Text feature [one] present in test data point [True]
76 Text feature [transfected] present in test data point [True]
79 Text feature [10] present in test data point [True]
80 Text feature [effects] present in test data point [True]
81 Text feature [mutation] present in test data point [True]
82 Text feature [mutants] present in test data point [True]
83 Text feature [reported] present in test data point [True]
87 Text feature [changes] present in test data point [True]
89 Text feature [expression] present in test data point [True]
90 Text feature [could] present in test data point [True]
91 Text feature [analyzed] present in test data point [True]
93 Text feature [resulting] present in test data point [True]
94 Text feature [possible] present in test data point [True]
96 Text feature [site] present in test data point [True]
98 Text feature [found] present in test data point [True]
99 Text feature [result] present in test data point [True]
Out of the top  100  features  59 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0823 0.0743 0.1383 0.0824 0.0467 0.0
468 0.519  0.0046 0.0055]]
Actual Class : 3
--------------------------------------------------
17 Text feature [activation] present in test data point [True]
18 Text feature [activated] present in test data point [True]
19 Text feature [kinase] present in test data point [True]
24 Text feature [inhibitor] present in test data point [True]
25 Text feature [cells] present in test data point [True]
26 Text feature [expressing] present in test data point [True]
28 Text feature [signaling] present in test data point [True]
29 Text feature [contrast] present in test data point [True]
30 Text feature [sensitive] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [also] present in test data point [True]
35 Text feature [however] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [activating] present in test data point [True]
38 Text feature [inhibitors] present in test data point [True]
39 Text feature [shown] present in test data point [True]
42 Text feature [addition] present in test data point [True]
43 Text feature [compared] present in test data point [True]
44 Text feature [presence] present in test data point [True]
45 Text feature [well] present in test data point [True]
```

```
46 Text feature [cell] present in test data point [True]
47 Text feature [previously] present in test data point [True]
48 Text feature [mutations] present in test data point [True]
49 Text feature [treatment] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [treated] present in test data point [True]
52 Text feature [similar] present in test data point [True]
53 Text feature [suggest] present in test data point [True]
54 Text feature [recently] present in test data point [True]
55 Text feature [mechanism] present in test data point [True]
56 Text feature [higher] present in test data point [True]
57 Text feature [inhibition] present in test data point [True]
58 Text feature [phosphorylation] present in test data point [True]
60 Text feature [oncogenic] present in test data point [True]
61 Text feature [found] present in test data point [True]
62 Text feature [3b] present in test data point [True]
67 Text feature [pathway] present in test data point [True]
68 Text feature [potential] present in test data point [True]
69 Text feature [induced] present in test data point [True]
71 Text feature [activate] present in test data point [True]
72 Text feature [pathways] present in test data point [True]
74 Text feature [increased] present in test data point [True]
75 Text feature [enhanced] present in test data point [True]
76 Text feature [results] present in test data point [True]
77 Text feature [proliferation] present in test data point [True]
78 Text feature [either] present in test data point [True]
79 Text feature [3a] present in test data point [True]
80 Text feature [culture] present in test data point [True]
81 Text feature [discussion] present in test data point [True]
82 Text feature [described] present in test data point [True]
83 Text feature [although] present in test data point [True]
84 Text feature [18] present in test data point [True]
85 Text feature [figure] present in test data point [True]
86 Text feature [observed] present in test data point [True]
87 Text feature [recent] present in test data point [True]
89 Text feature [mutation] present in test data point [True]
90 Text feature [12] present in test data point [True]
91 Text feature [respectively] present in test data point [True]
92 Text feature [mutant] present in test data point [True]
```

```
93 Text feature [trials] present in test data point [True]
94 Text feature [various] present in test data point [True]
95 Text feature [consistent] present in test data point [True]
96 Text feature [increase] present in test data point [True]
97 Text feature [including] present in test data point [True]
98 Text feature [reported] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

4.2. K Nearest Neighbour Classification

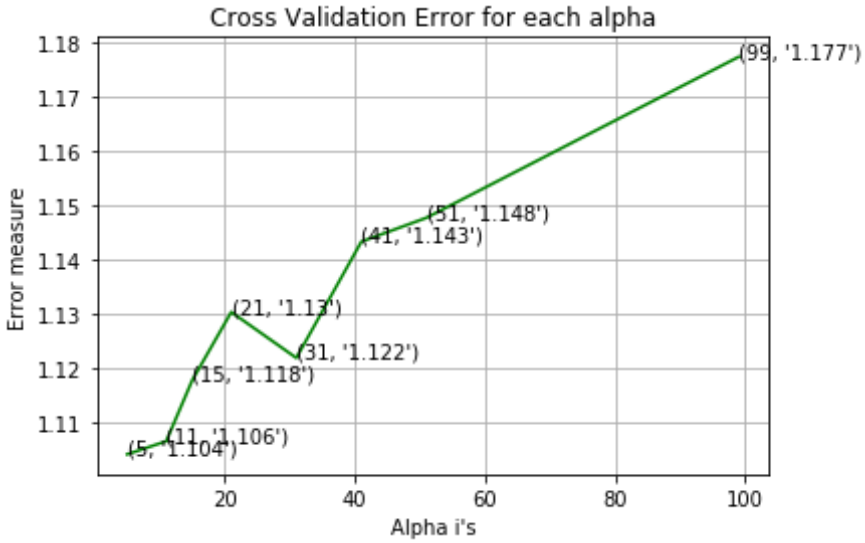
4.2.1. Hyper parameter tuning

In [0]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

```
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, 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 = [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 : 1.1039685507401267
for alpha = 11
Log Loss : 1.106380779753897
for alpha = 15
Log Loss : 1.1176390395447189
for alpha = 21
Log Loss : 1.1302383180522781
for alpha = 31
Log Loss : 1.1217667033866425
for alpha = 41
Log Loss : 1.1432436623294582
for alpha = 51
Log Loss : 1.1476495858669271
for alpha = 99
Log Loss : 1.1773746616879308
```



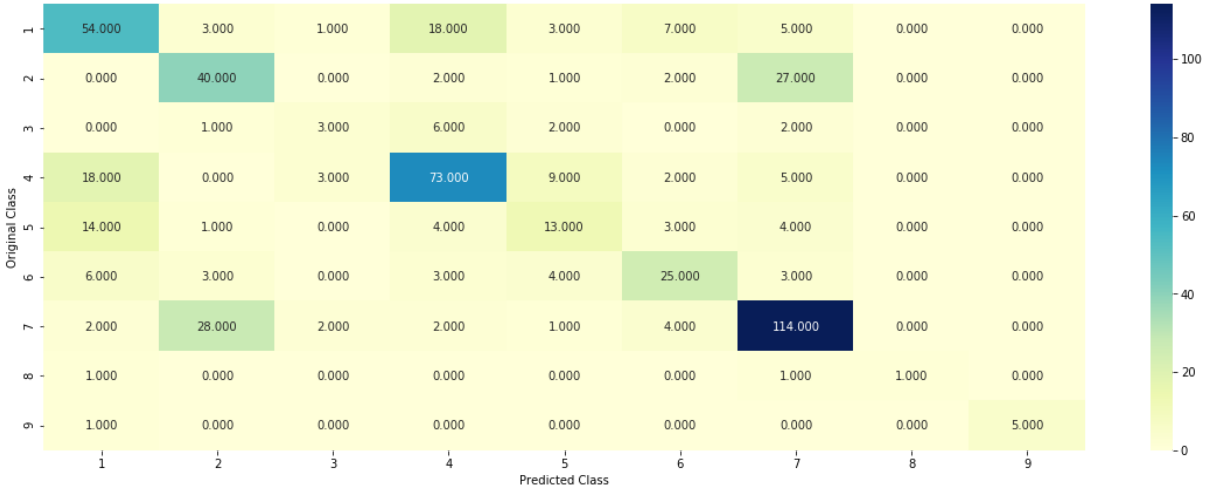

```
For values of best alpha =  5 The train log loss is: 0.4772188583588340
4
For values of best alpha =  5 The cross validation log loss is: 1.10396
85507401267
For values of best alpha =  5 The test log loss is: 1.1383043424426245
```

4.2.2. Testing the model with best hyper paramters

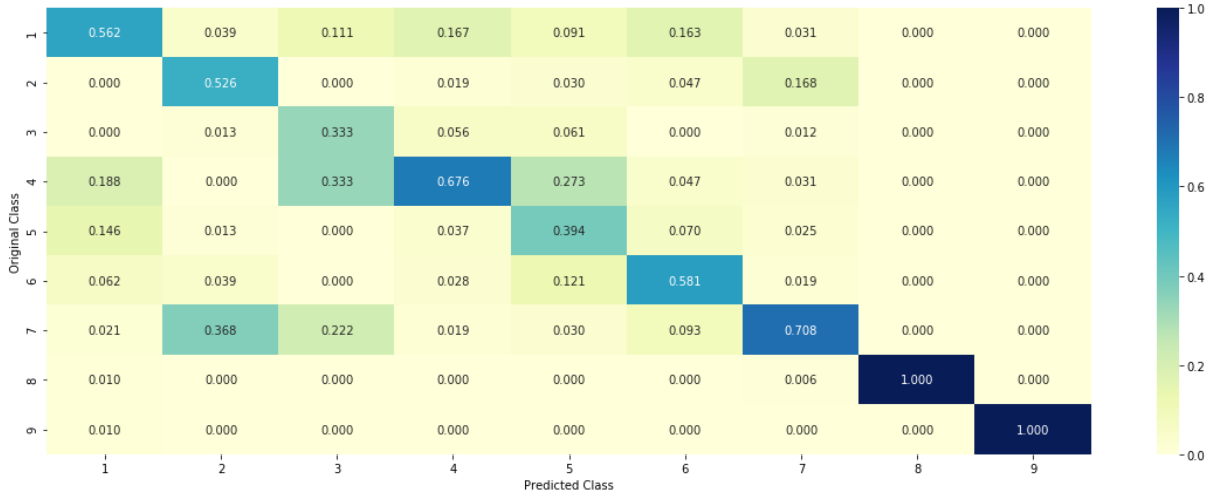
In [0]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', 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-examp
le-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x
_responseCoding, cv_y, clf)
```

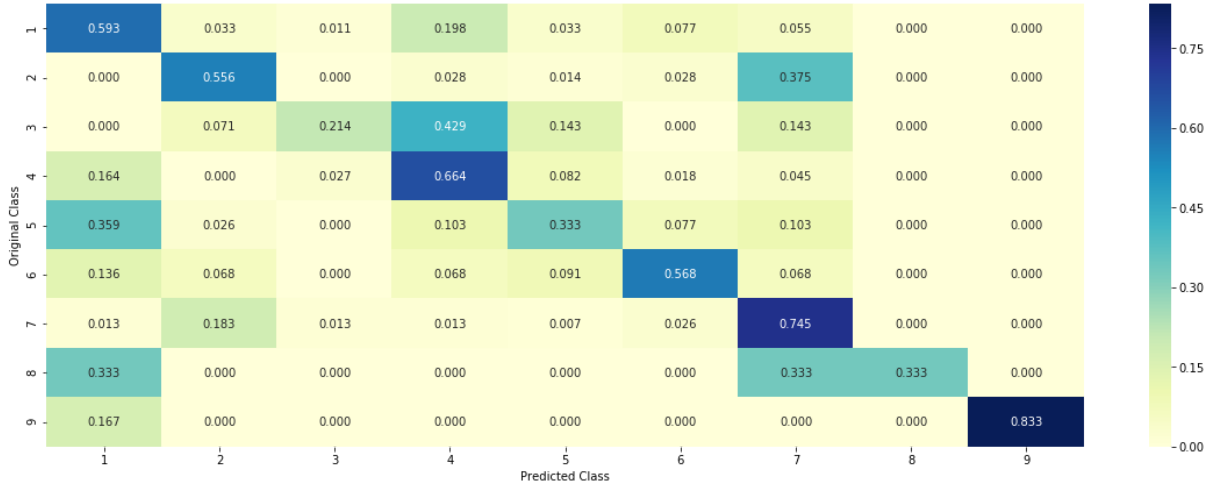
```
Log loss : 1.1039685507401267
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```




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



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



4.2.3.Sample Query point -1

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].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 : 1
Actual Class : 5
The  5  nearest neighbours of the test points belongs to classes [4 4 4
4 4]
Fequency of nearest points : Counter({4: 5})
```

4.2.4. Sample Query Point-2

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

test_point_index = 100
```

```
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].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 : 3
Actual Class : 3
the k value for knn is 5 and the nearest neighbours of the test points
belongs to classes [3 3 3 3 3]
Fequency of nearest points : Counter({3: 5})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

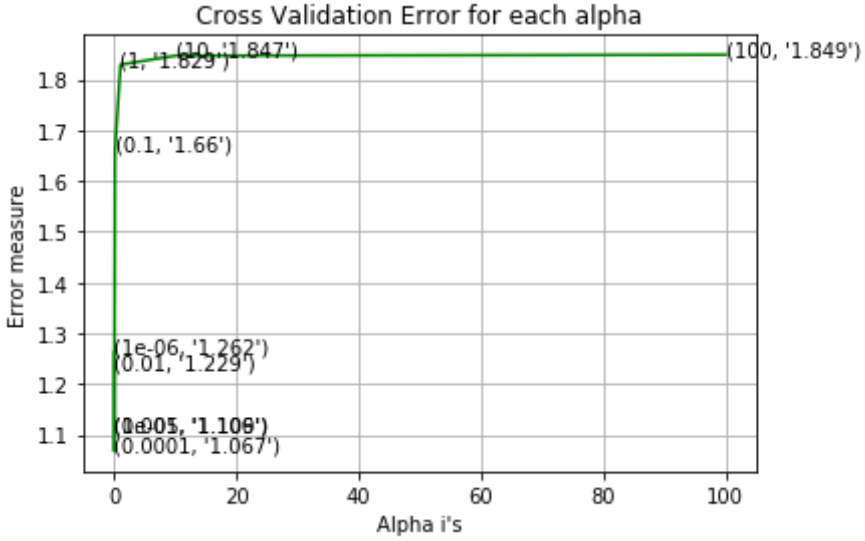
In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, 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 : 1.262181641483314
for alpha = 1e-05
Log Loss : 1.1089120891623376
for alpha = 0.0001
Log Loss : 1.0668751749577636
for alpha = 0.001
```

```
Log Loss : 1.1055975658387374
for alpha = 0.01
Log Loss : 1.2289868793691474
for alpha = 0.1

Log Loss : 1.6600817664625562
for alpha = 1
Log Loss : 1.8291616528687038
for alpha = 10
Log Loss : 1.8467979603847866
for alpha = 100
Log Loss : 1.8487344504590277
```



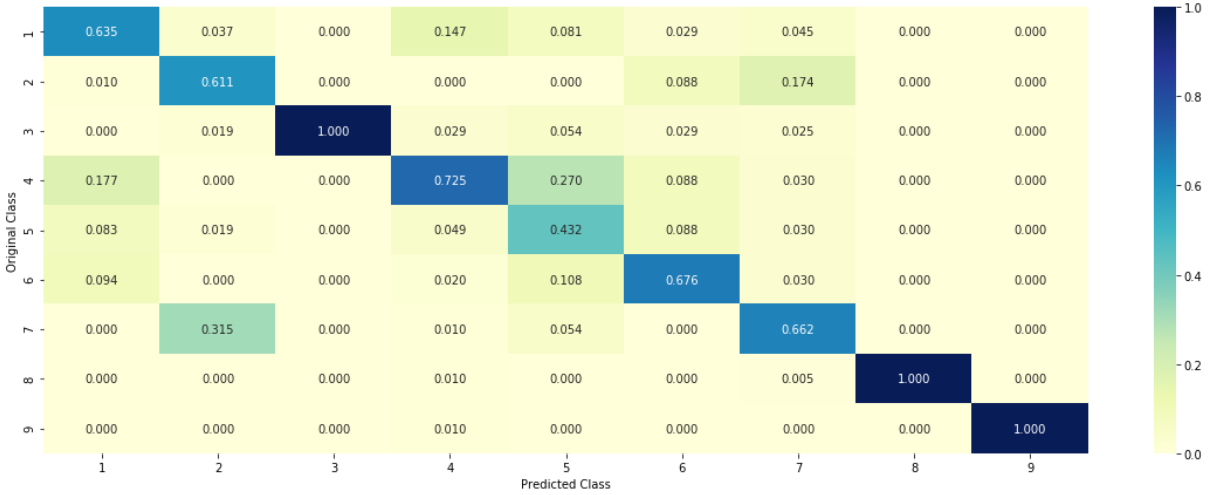

```
For values of best alpha =  0.0001 The train log loss is: 0.55589218452
10903
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0668751749577636
For values of best alpha =  0.0001 The test log loss is: 1.059586034891
8453
```

4.3.1.2. Testing the model with best hyper paramters

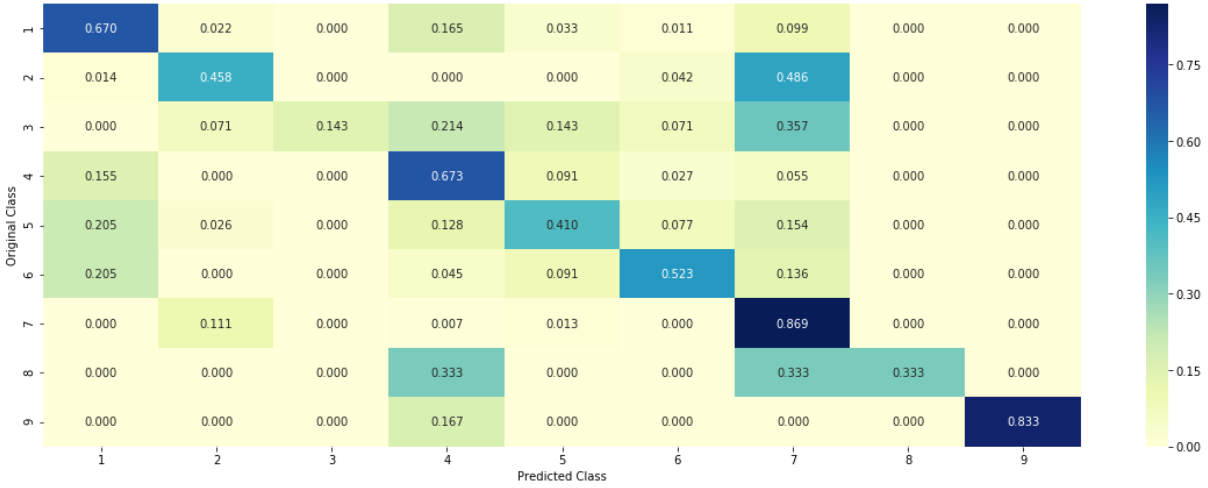
In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

```
Log loss : 1.0668751749577636
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```

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



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



4.3.1.3. Feature Importance

In [0]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_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 [0]:
```
# 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[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[1.830e-02 1.960e-02 1.750e-02 9.142e-0
1 4.600e-03 1.550e-02 7.900e-03
  1.600e-03 8.000e-04]]
Actual Class : 5
--------------------------------------------------
5 Text feature [suppressor] present in test data point [True]
31 Text feature [inactivation] present in test data point [True]
47 Text feature [damage] present in test data point [True]
76 Text feature [transfected] present in test data point [True]
86 Text feature [missense] present in test data point [True]
103 Text feature [defective] present in test data point [True]
144 Text feature [western] present in test data point [True]
```

```
169 Text feature [null] present in test data point [True]
178 Text feature [protein] present in test data point [True]
186 Text feature [resulting] present in test data point [True]
195 Text feature [leads] present in test data point [True]
201 Text feature [changes] present in test data point [True]
208 Text feature [repair] present in test data point [True]
216 Text feature [proportion] present in test data point [True]
219 Text feature [29] present in test data point [True]
220 Text feature [loss] present in test data point [True]
229 Text feature [age] present in test data point [True]
244 Text feature [alterations] present in test data point [True]
265 Text feature [bp] present in test data point [True]
270 Text feature [functional] present in test data point [True]
272 Text feature [similarly] present in test data point [True]
273 Text feature [high] present in test data point [True]
275 Text feature [cancers] present in test data point [True]
280 Text feature [direct] present in test data point [True]
281 Text feature [isolated] present in test data point [True]
286 Text feature [left] present in test data point [True]
287 Text feature [determine] present in test data point [True]
291 Text feature [phosphatase] present in test data point [True]
293 Text feature [cases] present in test data point [True]
305 Text feature [antibody] present in test data point [True]
307 Text feature [function] present in test data point [True]
311 Text feature [investigated] present in test data point [True]
315 Text feature [expected] present in test data point [True]
316 Text feature [suggesting] present in test data point [True]
325 Text feature [52] present in test data point [True]
340 Text feature [described] present in test data point [True]
351 Text feature [key] present in test data point [True]
352 Text feature [manufacturer] present in test data point [True]
353 Text feature [lines] present in test data point [True]
354 Text feature [splice] present in test data point [True]
360 Text feature [red] present in test data point [True]
361 Text feature [antibodies] present in test data point [True]
363 Text feature [normal] present in test data point [True]
364 Text feature [induction] present in test data point [True]
365 Text feature [blot] present in test data point [True]
366 Text feature [38] present in test data point [True]
```

```
368 Text feature [examined] present in test data point [True]
370 Text feature [ii] present in test data point [True]
374 Text feature [frequently] present in test data point [True]
377 Text feature [affinity] present in test data point [True]
378 Text feature [rate] present in test data point [True]
379 Text feature [mutants] present in test data point [True]
385 Text feature [predicted] present in test data point [True]
386 Text feature [altered] present in test data point [True]
391 Text feature [lesions] present in test data point [True]
394 Text feature [tagged] present in test data point [True]
395 Text feature [anti] present in test data point [True]
402 Text feature [complex] present in test data point [True]
411 Text feature [reduced] present in test data point [True]
417 Text feature [large] present in test data point [True]
420 Text feature [test] present in test data point [True]
430 Text feature [primary] present in test data point [True]
432 Text feature [long] present in test data point [True]
434 Text feature [200] present in test data point [True]
454 Text feature [involved] present in test data point [True]
455 Text feature [level] present in test data point [True]
456 Text feature [figure] present in test data point [True]
457 Text feature [indicates] present in test data point [True]
462 Text feature [12] present in test data point [True]
464 Text feature [thus] present in test data point [True]
466 Text feature [demonstrate] present in test data point [True]
471 Text feature [cultured] present in test data point [True]
478 Text feature [observed] present in test data point [True]
481 Text feature [process] present in test data point [True]
482 Text feature [ha] present in test data point [True]
484 Text feature [site] present in test data point [True]
488 Text feature [induced] present in test data point [True]
490 Text feature [since] present in test data point [True]
497 Text feature [therefore] present in test data point [True]
Out of the top  500  features  79 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [0]:
```
test_point_index = 100
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0098 0.0507 0.238  0.0168 0.0245 0.0
026 0.6538 0.0021 0.0017]]
Actual Class : 3
--------------------------------------------------
3 Text feature [codon] present in test data point [True]
6 Text feature [activation] present in test data point [True]
26 Text feature [activate] present in test data point [True]
27 Text feature [2a] present in test data point [True]
29 Text feature [activated] present in test data point [True]
38 Text feature [enhanced] present in test data point [True]
41 Text feature [sensitive] present in test data point [True]
43 Text feature [3b] present in test data point [True]
45 Text feature [signaling] present in test data point [True]
48 Text feature [presence] present in test data point [True]
65 Text feature [expressing] present in test data point [True]
68 Text feature [positive] present in test data point [True]
71 Text feature [pathways] present in test data point [True]
74 Text feature [stably] present in test data point [True]
82 Text feature [mechanism] present in test data point [True]
83 Text feature [phospho] present in test data point [True]
91 Text feature [inhibitor] present in test data point [True]
92 Text feature [culture] present in test data point [True]
105 Text feature [membrane] present in test data point [True]
106 Text feature [approximately] present in test data point [True]
111 Text feature [leading] present in test data point [True]
115 Text feature [days] present in test data point [True]
142 Text feature [regulated] present in test data point [True]
```

```
144 Text feature [oncogenic] present in test data point [True]
151 Text feature [activating] present in test data point [True]
153 Text feature [per] present in test data point [True]
174 Text feature [mechanisms] present in test data point [True]
175 Text feature [level] present in test data point [True]
191 Text feature [pi3k] present in test data point [True]
231 Text feature [lead] present in test data point [True]
233 Text feature [phosphorylation] present in test data point [True]
249 Text feature [increased] present in test data point [True]
255 Text feature [pathway] present in test data point [True]
266 Text feature [high] present in test data point [True]
273 Text feature [increase] present in test data point [True]
275 Text feature [upon] present in test data point [True]
278 Text feature [express] present in test data point [True]
279 Text feature [3a] present in test data point [True]
286 Text feature [responses] present in test data point [True]
291 Text feature [distinct] present in test data point [True]
292 Text feature [24] present in test data point [True]
293 Text feature [mutant] present in test data point [True]
294 Text feature [occur] present in test data point [True]
316 Text feature [additional] present in test data point [True]
319 Text feature [size] present in test data point [True]
322 Text feature [survival] present in test data point [True]
331 Text feature [akt] present in test data point [True]
348 Text feature [18] present in test data point [True]
351 Text feature [addition] present in test data point [True]
356 Text feature [suggest] present in test data point [True]
367 Text feature [either] present in test data point [True]
369 Text feature [2000] present in test data point [True]
370 Text feature [cultured] present in test data point [True]
377 Text feature [akt1] present in test data point [True]
380 Text feature [mutants] present in test data point [True]
381 Text feature [cdna] present in test data point [True]
384 Text feature [possible] present in test data point [True]
385 Text feature [resulting] present in test data point [True]
388 Text feature [day] present in test data point [True]
389 Text feature [found] present in test data point [True]
391 Text feature [obtained] present in test data point [True]
396 Text feature [12] present in test data point [True]
411 Text feature [2b] present in test data point [True]
```

```
412 Text feature [contrast] present in test data point [True]
414 Text feature [tumor] present in test data point [True]
416 Text feature [ph] present in test data point [True]
417 Text feature [lung] present in test data point [True]
428 Text feature [kinase] present in test data point [True]
434 Text feature [supplemental] present in test data point [True]
436 Text feature [strongly] present in test data point [True]
438 Text feature [growth] present in test data point [True]
439 Text feature [free] present in test data point [True]
441 Text feature [interestingly] present in test data point [True]
444 Text feature [previously] present in test data point [True]
447 Text feature [involving] present in test data point [True]
451 Text feature [various] present in test data point [True]
452 Text feature [often] present in test data point [True]
455 Text feature [induced] present in test data point [True]
456 Text feature [led] present in test data point [True]
457 Text feature [sensitivity] present in test data point [True]
463 Text feature [cells] present in test data point [True]
466 Text feature [figures] present in test data point [True]
474 Text feature [19] present in test data point [True]
476 Text feature [stimulation] present in test data point [True]
477 Text feature [consistent] present in test data point [True]
478 Text feature [recently] present in test data point [True]
479 Text feature [examined] present in test data point [True]
481 Text feature [proliferation] present in test data point [True]
485 Text feature [kras] present in test data point [True]
499 Text feature [72] present in test data point [True]
Out of the top  500  features  90 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

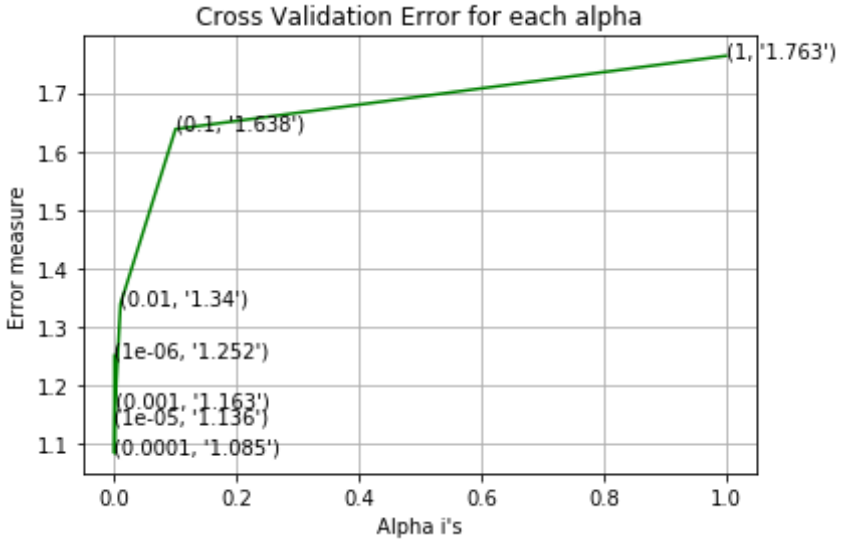
In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
```

```
# ------------------------------
# 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 : 1.2517367877633938
for alpha = 1e-05
Log Loss : 1.13591609201107
for alpha = 0.0001
Log Loss : 1.0847711857361417
for alpha = 0.001
Log Loss : 1.1630591324133384
for alpha = 0.01
Log Loss : 1.3397702346104183
for alpha = 0.1
Log Loss : 1.638028510468913
for alpha = 1
Log Loss : 1.7630154522041097
```



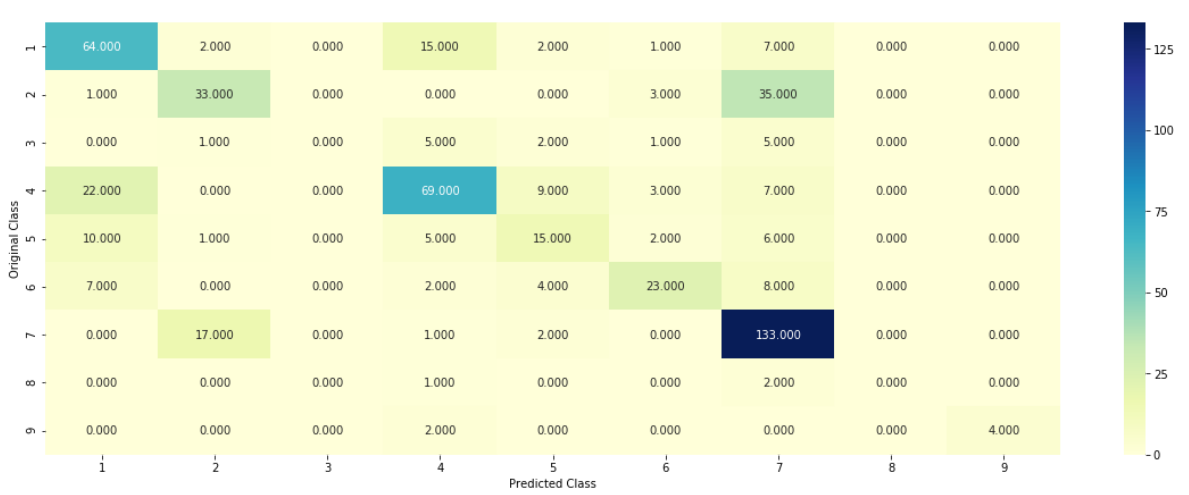

```
For values of best alpha =  0.0001 The train log loss is: 0.5534125966965848
For values of best alpha =  0.0001 The cross validation log loss is: 1.0847711857361417
For values of best alpha =  0.0001 The test log loss is: 1.0807995586747963
```

7903

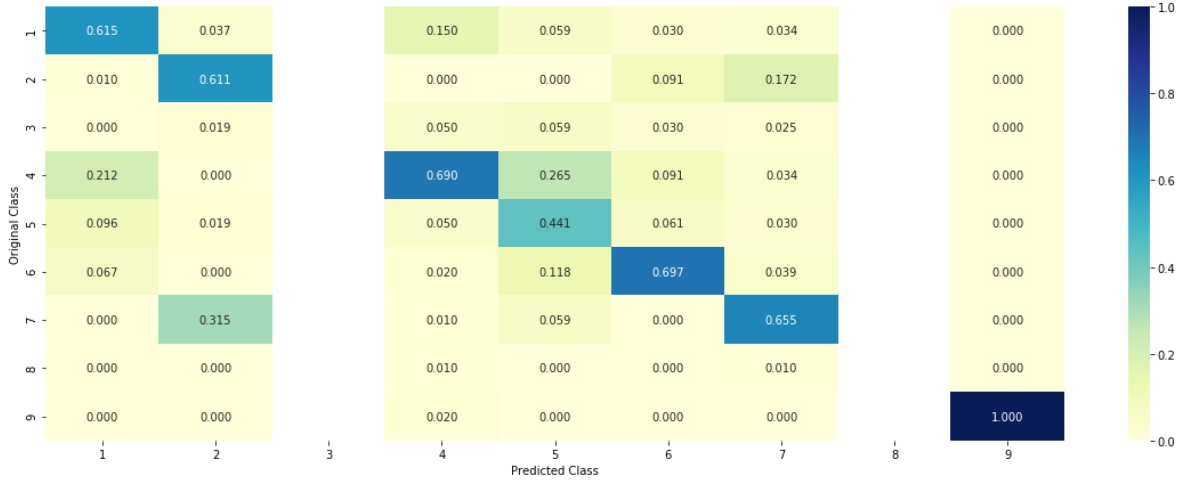
4.3.2.2. Testing model with best hyper parameters

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

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

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

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

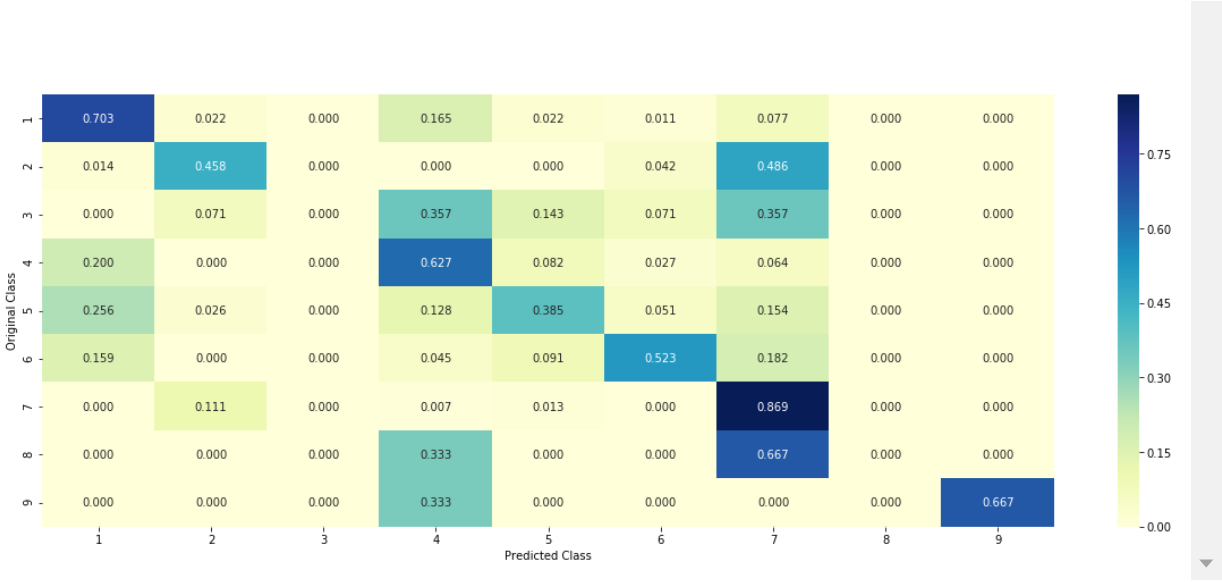
```
Log loss : 1.0847711857361417
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

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

```
Predicted Class : 4
Predicted Class Probabilities: [[2.030e-02 2.090e-02 1.060e-02 9.163e-0
1 5.000e-03 1.440e-02 1.080e-02
```

```
  1.300e-03 4.000e-04]]

Actual Class : 5
--------------------------------------------------
8 Text feature [suppressor] present in test data point [True]
43 Text feature [damage] present in test data point [True]
48 Text feature [inactivation] present in test data point [True]
62 Text feature [defective] present in test data point [True]
86 Text feature [transfected] present in test data point [True]
107 Text feature [western] present in test data point [True]
109 Text feature [missense] present in test data point [True]
145 Text feature [leads] present in test data point [True]
153 Text feature [resulting] present in test data point [True]
171 Text feature [protein] present in test data point [True]
173 Text feature [repair] present in test data point [True]
175 Text feature [29] present in test data point [True]
183 Text feature [proportion] present in test data point [True]
190 Text feature [age] present in test data point [True]
191 Text feature [similarly] present in test data point [True]
208 Text feature [cancers] present in test data point [True]
219 Text feature [loss] present in test data point [True]
222 Text feature [null] present in test data point [True]
235 Text feature [changes] present in test data point [True]
254 Text feature [high] present in test data point [True]
263 Text feature [functional] present in test data point [True]
269 Text feature [alterations] present in test data point [True]
272 Text feature [function] present in test data point [True]
279 Text feature [cases] present in test data point [True]
281 Text feature [determine] present in test data point [True]
282 Text feature [bp] present in test data point [True]
283 Text feature [left] present in test data point [True]
293 Text feature [antibody] present in test data point [True]
294 Text feature [isolated] present in test data point [True]
298 Text feature [investigated] present in test data point [True]
300 Text feature [described] present in test data point [True]
306 Text feature [key] present in test data point [True]
312 Text feature [lines] present in test data point [True]
313 Text feature [expected] present in test data point [True]
314 Text feature [manufacturer] present in test data point [True]
316 Text feature [52] present in test data point [True]
```

```
318 Text feature [examined] present in test data point [True]

319 Text feature [direct] present in test data point [True]
325 Text feature [red] present in test data point [True]
327 Text feature [induction] present in test data point [True]
334 Text feature [phosphatase] present in test data point [True]
340 Text feature [blot] present in test data point [True]
341 Text feature [long] present in test data point [True]
344 Text feature [38] present in test data point [True]
349 Text feature [splice] present in test data point [True]
354 Text feature [predicted] present in test data point [True]
363 Text feature [anti] present in test data point [True]
365 Text feature [suggesting] present in test data point [True]
368 Text feature [normal] present in test data point [True]
378 Text feature [affinity] present in test data point [True]
381 Text feature [tagged] present in test data point [True]
387 Text feature [antibodies] present in test data point [True]
388 Text feature [ha] present in test data point [True]
389 Text feature [rate] present in test data point [True]
400 Text feature [thus] present in test data point [True]
402 Text feature [large] present in test data point [True]
404 Text feature [indicates] present in test data point [True]
410 Text feature [ii] present in test data point [True]
413 Text feature [observed] present in test data point [True]
418 Text feature [primary] present in test data point [True]
420 Text feature [reduced] present in test data point [True]
421 Text feature [lesions] present in test data point [True]
423 Text feature [mutants] present in test data point [True]
425 Text feature [significantly] present in test data point [True]
426 Text feature [altered] present in test data point [True]
427 Text feature [frequently] present in test data point [True]
435 Text feature [figure] present in test data point [True]
436 Text feature [site] present in test data point [True]
437 Text feature [major] present in test data point [True]
438 Text feature [process] present in test data point [True]
439 Text feature [test] present in test data point [True]
442 Text feature [control] present in test data point [True]
445 Text feature [induced] present in test data point [True]
448 Text feature [200] present in test data point [True]
450 Text feature [cultured] present in test data point [True]
```

```
462 Text feature [level] present in test data point [True]

464 Text feature [remaining] present in test data point [True]
475 Text feature [often] present in test data point [True]
476 Text feature [treated] present in test data point [True]
483 Text feature [complex] present in test data point [True]
487 Text feature [mutagenesis] present in test data point [True]
493 Text feature [12] present in test data point [True]
499 Text feature [therefore] present in test data point [True]
Out of the top  500  features  83 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0114 0.0536 0.1597 0.0198 0.024  0.0
027 0.7262 0.0016 0.001 ]]
Actual Class : 3
--------------------------------------------------
5 Text feature [codon] present in test data point [True]
13 Text feature [activation] present in test data point [True]
24 Text feature [activate] present in test data point [True]
31 Text feature [sensitive] present in test data point [True]
34 Text feature [2a] present in test data point [True]
39 Text feature [3b] present in test data point [True]
40 Text feature [enhanced] present in test data point [True]
44 Text feature [presence] present in test data point [True]
```

```
51 Text feature [positive] present in test data point [True]
61 Text feature [activated] present in test data point [True]
69 Text feature [signaling] present in test data point [True]
81 Text feature [stably] present in test data point [True]
83 Text feature [pathways] present in test data point [True]
87 Text feature [approximately] present in test data point [True]
89 Text feature [expressing] present in test data point [True]
91 Text feature [inhibitor] present in test data point [True]
95 Text feature [membrane] present in test data point [True]
97 Text feature [mechanism] present in test data point [True]
108 Text feature [phospho] present in test data point [True]
109 Text feature [leading] present in test data point [True]
110 Text feature [culture] present in test data point [True]
121 Text feature [regulated] present in test data point [True]
127 Text feature [days] present in test data point [True]
137 Text feature [per] present in test data point [True]
140 Text feature [activating] present in test data point [True]
152 Text feature [level] present in test data point [True]
181 Text feature [high] present in test data point [True]
185 Text feature [increased] present in test data point [True]
201 Text feature [additional] present in test data point [True]
204 Text feature [lead] present in test data point [True]
209 Text feature [distinct] present in test data point [True]
211 Text feature [mechanisms] present in test data point [True]
221 Text feature [upon] present in test data point [True]
227 Text feature [responses] present in test data point [True]
233 Text feature [pi3k] present in test data point [True]
247 Text feature [size] present in test data point [True]
249 Text feature [phosphorylation] present in test data point [True]
258 Text feature [mutant] present in test data point [True]
259 Text feature [possible] present in test data point [True]
262 Text feature [oncogenic] present in test data point [True]
263 Text feature [increase] present in test data point [True]
283 Text feature [ph] present in test data point [True]
292 Text feature [24] present in test data point [True]
294 Text feature [suggest] present in test data point [True]
295 Text feature [express] present in test data point [True]
299 Text feature [day] present in test data point [True]
311 Text feature [3a] present in test data point [True]
```

```
313 Text feature [resulting] present in test data point [True]
314 Text feature [pathway] present in test data point [True]
316 Text feature [addition] present in test data point [True]
318 Text feature [2000] present in test data point [True]
327 Text feature [18] present in test data point [True]
330 Text feature [either] present in test data point [True]
354 Text feature [akt] present in test data point [True]
361 Text feature [mutants] present in test data point [True]
367 Text feature [akt1] present in test data point [True]
368 Text feature [survival] present in test data point [True]
376 Text feature [19] present in test data point [True]
378 Text feature [cdna] present in test data point [True]
380 Text feature [obtained] present in test data point [True]
381 Text feature [found] present in test data point [True]
386 Text feature [contrast] present in test data point [True]
387 Text feature [recently] present in test data point [True]
388 Text feature [cultured] present in test data point [True]
406 Text feature [led] present in test data point [True]
418 Text feature [occur] present in test data point [True]
427 Text feature [figures] present in test data point [True]
429 Text feature [free] present in test data point [True]
431 Text feature [tumor] present in test data point [True]
432 Text feature [2b] present in test data point [True]
434 Text feature [previously] present in test data point [True]
435 Text feature [negative] present in test data point [True]
441 Text feature [sensitivity] present in test data point [True]
442 Text feature [strongly] present in test data point [True]
444 Text feature [various] present in test data point [True]
445 Text feature [consistent] present in test data point [True]
451 Text feature [taken] present in test data point [True]
458 Text feature [interestingly] present in test data point [True]
459 Text feature [12] present in test data point [True]
460 Text feature [supplemental] present in test data point [True]
462 Text feature [materials] present in test data point [True]
469 Text feature [lung] present in test data point [True]
470 Text feature [kinase] present in test data point [True]
476 Text feature [kras] present in test data point [True]
484 Text feature [somatic] present in test data point [True]
485 Text feature [wt] present in test data point [True]
```

```
489 Text feature [often] present in test data point [True]
493 Text feature [stimulated] present in test data point [True]
494 Text feature [involving] present in test data point [True]
495 Text feature [growth] present in test data point [True]
497 Text feature [subjected] present in test data point [True]
499 Text feature [examined] present in test data point [True]
Out of the top  500  features  92 are present in query point
```

4.4. Linear Support Vector Machines

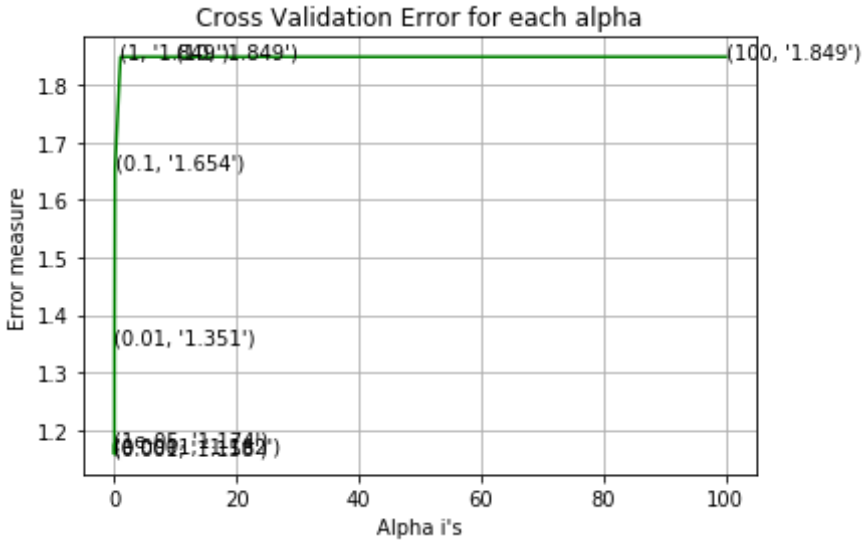
4.4.1. Hyper paramter tuning

In [0]:
```
# read more about support vector machines with linear kernals here 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 : 1.174147301703329
for C = 0.0001
Log Loss : 1.1622581360179502
for C = 0.001
Log Loss : 1.157685281395595
for C = 0.01
Log Loss : 1.3510324752912377
for C = 0.1
Log Loss : 1.6544875529205982
for C = 1
Log Loss : 1.8491177142157131
for C = 10
Log Loss : 1.84911703242191
for C = 100
Log Loss : 1.8491098766470504
```



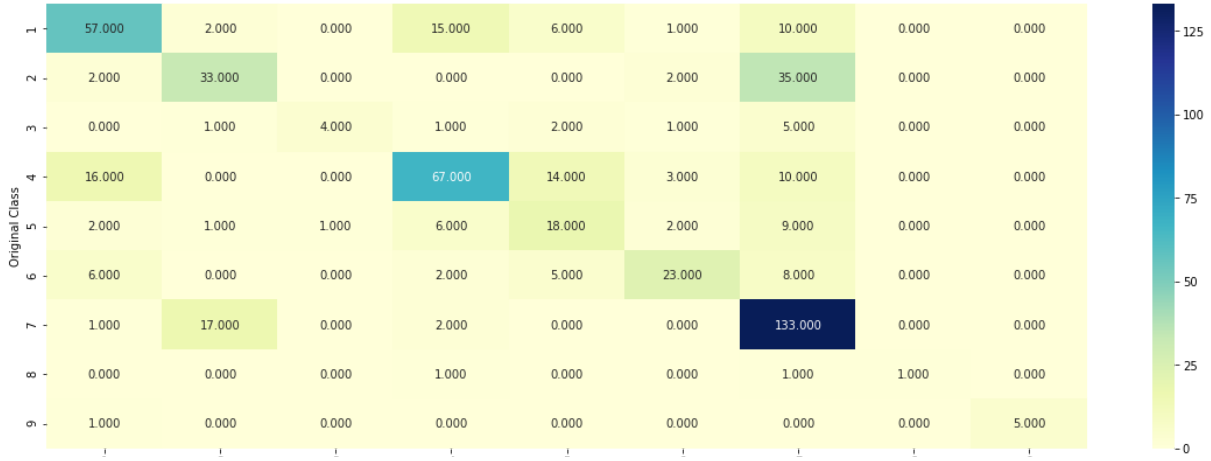

```
For values of best alpha =  0.001 The train log loss is: 0.7893184207
314842
For values of best alpha =  0.001 The cross validation log loss is:
1.157685281395595
For values of best alpha =  0.001 The test log loss is: 1.18884440017
52025
```

4.4.2. Testing model with best hyper parameters

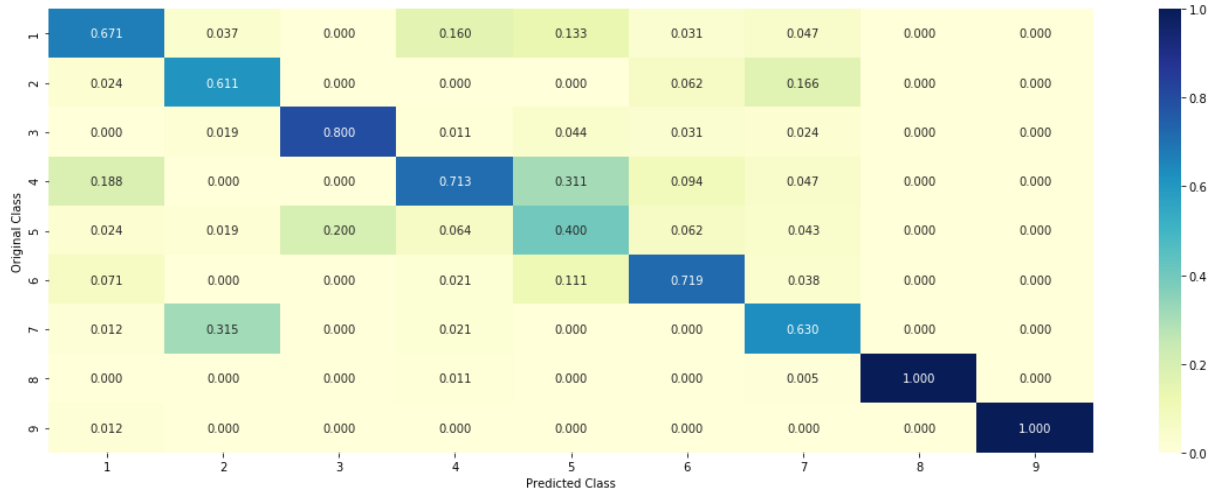
In [0]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

```
# 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)
```

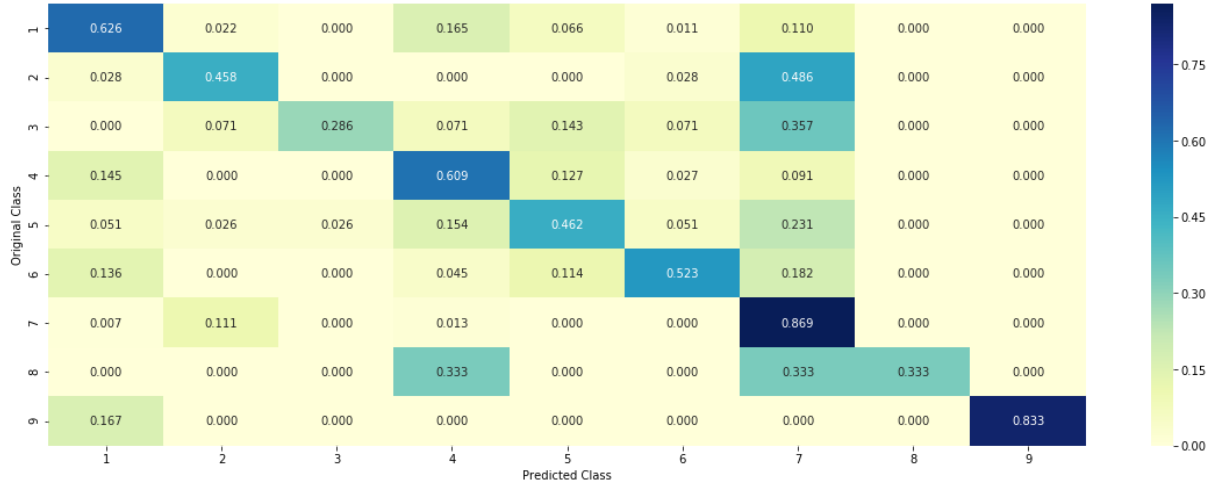
```
Log loss : 1.157685281395595
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0533 0.0591 0.01   0.7469 0.015  0.0
434 0.0687 0.002  0.0017]]
Actual Class : 5
--------------------------------------------------
11 Text feature [suppressor] present in test data point [True]
18 Text feature [inactivation] present in test data point [True]
25 Text feature [damage] present in test data point [True]
39 Text feature [transfected] present in test data point [True]
140 Text feature [proportion] present in test data point [True]
163 Text feature [missense] present in test data point [True]
165 Text feature [direct] present in test data point [True]
176 Text feature [alterations] present in test data point [True]
178 Text feature [changes] present in test data point [True]
183 Text feature [mutants] present in test data point [True]
184 Text feature [null] present in test data point [True]
185 Text feature [suggesting] present in test data point [True]
199 Text feature [resulting] present in test data point [True]
```

```
199 Text feature [resulting] present in test data point [True]
201 Text feature [western] present in test data point [True]

213 Text feature [rate] present in test data point [True]
216 Text feature [29] present in test data point [True]
225 Text feature [complex] present in test data point [True]
234 Text feature [isolated] present in test data point [True]
236 Text feature [expected] present in test data point [True]
237 Text feature [tagged] present in test data point [True]
242 Text feature [functional] present in test data point [True]
245 Text feature [ii] present in test data point [True]
246 Text feature [repair] present in test data point [True]
247 Text feature [loss] present in test data point [True]
258 Text feature [high] present in test data point [True]
260 Text feature [splice] present in test data point [True]
261 Text feature [age] present in test data point [True]
262 Text feature [left] present in test data point [True]
268 Text feature [12] present in test data point [True]
286 Text feature [lesions] present in test data point [True]
288 Text feature [leads] present in test data point [True]
291 Text feature [cases] present in test data point [True]
292 Text feature [antibody] present in test data point [True]
298 Text feature [frequently] present in test data point [True]
304 Text feature [large] present in test data point [True]
318 Text feature [cancers] present in test data point [True]
322 Text feature [52] present in test data point [True]
323 Text feature [predicted] present in test data point [True]
325 Text feature [involved] present in test data point [True]
330 Text feature [determine] present in test data point [True]
331 Text feature [antibodies] present in test data point [True]
332 Text feature [38] present in test data point [True]
336 Text feature [described] present in test data point [True]
338 Text feature [bp] present in test data point [True]
341 Text feature [protein] present in test data point [True]
343 Text feature [lines] present in test data point [True]
346 Text feature [demonstrate] present in test data point [True]
349 Text feature [lanes] present in test data point [True]
351 Text feature [key] present in test data point [True]
364 Text feature [long] present in test data point [True]
368 Text feature [patient] present in test data point [True]
375 Text feature [since] present in test data point [True]
```

```
376 Text feature [indicates] present in test data point [True]

377 Text feature [previously] present in test data point [True]
382 Text feature [anti] present in test data point [True]
383 Text feature [associated] present in test data point [True]
387 Text feature [normal] present in test data point [True]
392 Text feature [red] present in test data point [True]
393 Text feature [23] present in test data point [True]
402 Text feature [primary] present in test data point [True]
403 Text feature [general] present in test data point [True]
405 Text feature [investigated] present in test data point [True]
406 Text feature [still] present in test data point [True]
408 Text feature [examined] present in test data point [True]
409 Text feature [case] present in test data point [True]
412 Text feature [analysis] present in test data point [True]
414 Text feature [s1] present in test data point [True]
416 Text feature [phosphatase] present in test data point [True]
420 Text feature [therefore] present in test data point [True]
422 Text feature [mutagenesis] present in test data point [True]
423 Text feature [new] present in test data point [True]
424 Text feature [site] present in test data point [True]
425 Text feature [often] present in test data point [True]
428 Text feature [figure] present in test data point [True]
430 Text feature [manufacturer] present in test data point [True]
442 Text feature [function] present in test data point [True]
443 Text feature [remaining] present in test data point [True]
444 Text feature [altered] present in test data point [True]
445 Text feature [amplified] present in test data point [True]
452 Text feature [1a] present in test data point [True]
454 Text feature [genomic] present in test data point [True]
458 Text feature [200] present in test data point [True]
462 Text feature [reduced] present in test data point [True]
465 Text feature [induced] present in test data point [True]
471 Text feature [amino] present in test data point [True]
472 Text feature [mutated] present in test data point [True]
473 Text feature [36] present in test data point [True]
474 Text feature [observed] present in test data point [True]
478 Text feature [defective] present in test data point [True]
484 Text feature [50] present in test data point [True]
488 Text feature [evaluated] present in test data point [True]
```

```
497 Text feature [lead] present in test data point [True]

498 Text feature [used] present in test data point [True]
Out of the top  500  features  93 are present in query point
```

4.3.3.2. For Incorrectly classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0745 0.0779 0.1875 0.1225 0.0434 0.0
171 0.4712 0.0026 0.0032]]
Actual Class : 3
--------------------------------------------------
6 Text feature [codon] present in test data point [True]
15 Text feature [2a] present in test data point [True]
22 Text feature [activation] present in test data point [True]
25 Text feature [activated] present in test data point [True]
26 Text feature [sensitive] present in test data point [True]
27 Text feature [activate] present in test data point [True]
43 Text feature [membrane] present in test data point [True]
44 Text feature [3b] present in test data point [True]
47 Text feature [signaling] present in test data point [True]
53 Text feature [positive] present in test data point [True]
54 Text feature [enhanced] present in test data point [True]
56 Text feature [expressing] present in test data point [True]
57 Text feature [inhibitor] present in test data point [True]
62 Text feature [presence] present in test data point [True]
```

62 Text feature [presence] present in test data point [True]
64 Text feature [approximately] present in test data point [True]
66 Text feature [mechanism] present in test data point [True]
69 Text feature [pathways] present in test data point [True]
72 Text feature [culture] present in test data point [True]
76 Text feature [regulated] present in test data point [True]
79 Text feature [akt] present in test data point [True]
80 Text feature [stimulation] present in test data point [True]
82 Text feature [activating] present in test data point [True]
146 Text feature [lead] present in test data point [True]
147 Text feature [pi3k] present in test data point [True]
188 Text feature [phospho] present in test data point [True]
189 Text feature [akt1] present in test data point [True]
191 Text feature [level] present in test data point [True]
205 Text feature [mutants] present in test data point [True]
209 Text feature [mutant] present in test data point [True]
221 Text feature [suggest] present in test data point [True]
223 Text feature [pathway] present in test data point [True]
226 Text feature [stably] present in test data point [True]
227 Text feature [phosphorylation] present in test data point [True]
229 Text feature [leading] present in test data point [True]
263 Text feature [oncogenic] present in test data point [True]
264 Text feature [2b] present in test data point [True]
266 Text feature [addition] present in test data point [True]
267 Text feature [mechanisms] present in test data point [True]
270 Text feature [ph] present in test data point [True]
273 Text feature [18] present in test data point [True]
276 Text feature [additional] present in test data point [True]
290 Text feature [proliferation] present in test data point [True]
291 Text feature [per] present in test data point [True]
295 Text feature [express] present in test data point [True]
297 Text feature [p110] present in test data point [True]
298 Text feature [increased] present in test data point [True]
313 Text feature [hours] present in test data point [True]
320 Text feature [found] present in test data point [True]
332 Text feature [either] present in test data point [True]
334 Text feature [responses] present in test data point [True]
335 Text feature [survival] present in test data point [True]
337 Text feature [involving] present in test data point [True]
340 Text feature [24] present in test data point [True]

```
346 Text feature [days] present in test data point [True]
348 Text feature [2000] present in test data point [True]
364 Text feature [cultured] present in test data point [True]
365 Text feature [high] present in test data point [True]
366 Text feature [mtor] present in test data point [True]
370 Text feature [growth] present in test data point [True]
371 Text feature [lung] present in test data point [True]
372 Text feature [cells] present in test data point [True]
373 Text feature [sensitivity] present in test data point [True]
374 Text feature [obtained] present in test data point [True]
380 Text feature [stimulated] present in test data point [True]
382 Text feature [supplemental] present in test data point [True]
391 Text feature [increase] present in test data point [True]
392 Text feature [figures] present in test data point [True]
395 Text feature [3a] present in test data point [True]
398 Text feature [wt] present in test data point [True]
405 Text feature [primers] present in test data point [True]
407 Text feature [upon] present in test data point [True]
410 Text feature [tumor] present in test data point [True]
412 Text feature [12] present in test data point [True]
414 Text feature [taken] present in test data point [True]
415 Text feature [caused] present in test data point [True]
416 Text feature [negative] present in test data point [True]
418 Text feature [domain] present in test data point [True]
419 Text feature [inhibition] present in test data point [True]
421 Text feature [cdna] present in test data point [True]
423 Text feature [occur] present in test data point [True]
425 Text feature [patient] present in test data point [True]
427 Text feature [targeting] present in test data point [True]
428 Text feature [incubated] present in test data point [True]
431 Text feature [19] present in test data point [True]
444 Text feature [higher] present in test data point [True]
445 Text feature [recently] present in test data point [True]
452 Text feature [previously] present in test data point [True]
454 Text feature [possible] present in test data point [True]
455 Text feature [led] present in test data point [True]
456 Text feature [examined] present in test data point [True]
457 Text feature [plates] present in test data point [True]
461 Text feature [distinct] present in test data point [True]
```

```
474 Text feature [kinase] present in test data point [True]
475 Text feature [contrast] present in test data point [True]
478 Text feature [consistent] present in test data point [True]
485 Text feature [patients] present in test data point [True]
486 Text feature [various] present in test data point [True]
488 Text feature [free] present in test data point [True]
489 Text feature [resulting] present in test data point [True]
490 Text feature [may] present in test data point [True]
Out of the top  500  features  100 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [0]:
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='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.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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

```
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2321624631050092
for n_estimators = 100 and max depth =  10
Log Loss : 1.2495775757807577
for n_estimators = 200 and max depth =  5
Log Loss : 1.2180993373822575
```

```
for n_estimators = 200 and max depth =  10
Log Loss : 1.2435553325390227
for n_estimators = 500 and max depth =  5
Log Loss : 1.2087638437450443
for n_estimators = 500 and max depth =  10
Log Loss : 1.2378627269616274
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2081027665201598
for n_estimators = 1000 and max depth =  10

Log Loss : 1.2346927940531998
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2080198666205069
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2316848518113366
For values of best estimator =  2000 The train log loss is: 0.843210055
8401556
For values of best estimator =  2000 The cross validation log loss is:
1.2080198666205069
For values of best estimator =  2000 The test log loss is: 1.2544436898
616733
```

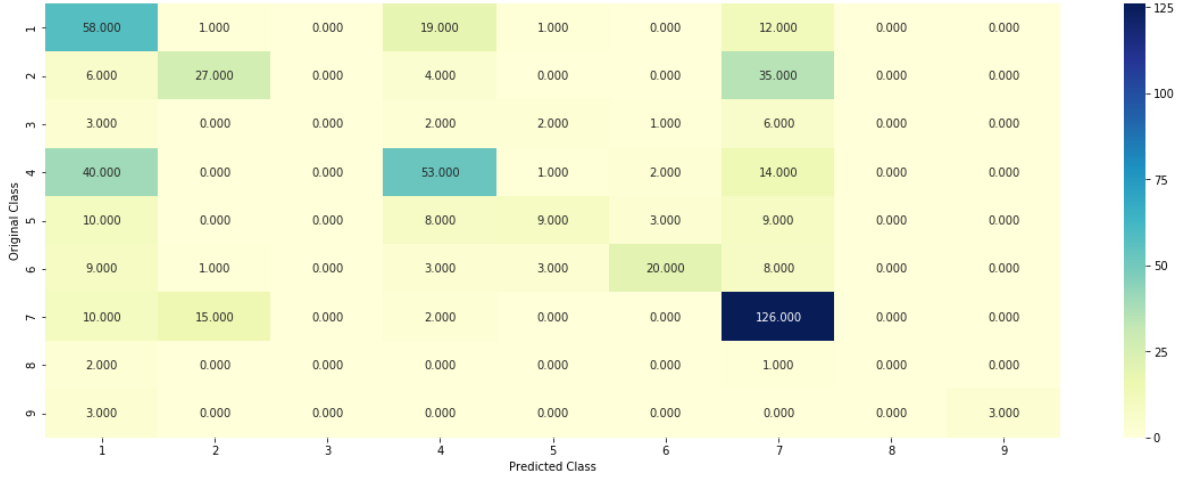
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [0]:

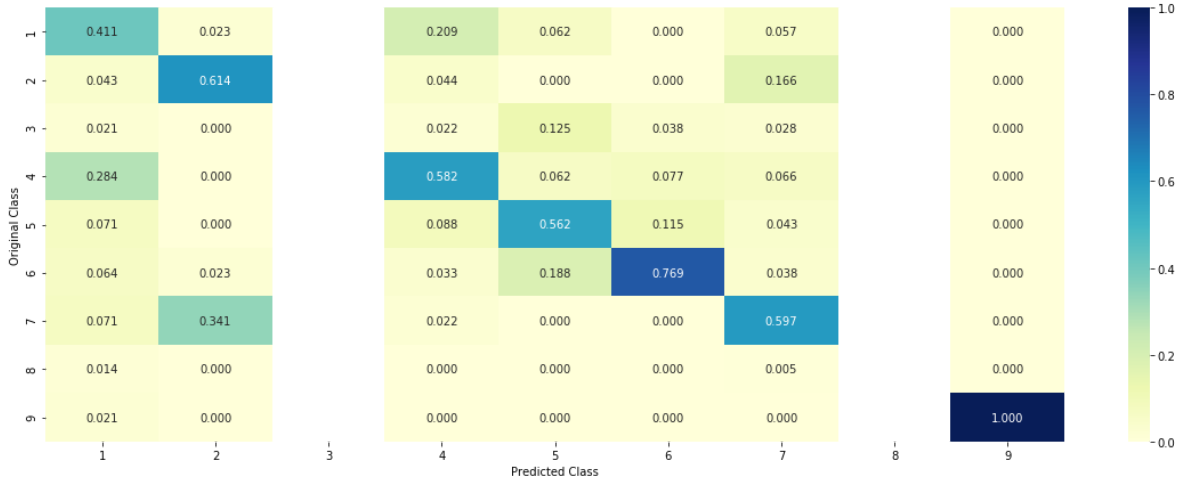
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, 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)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

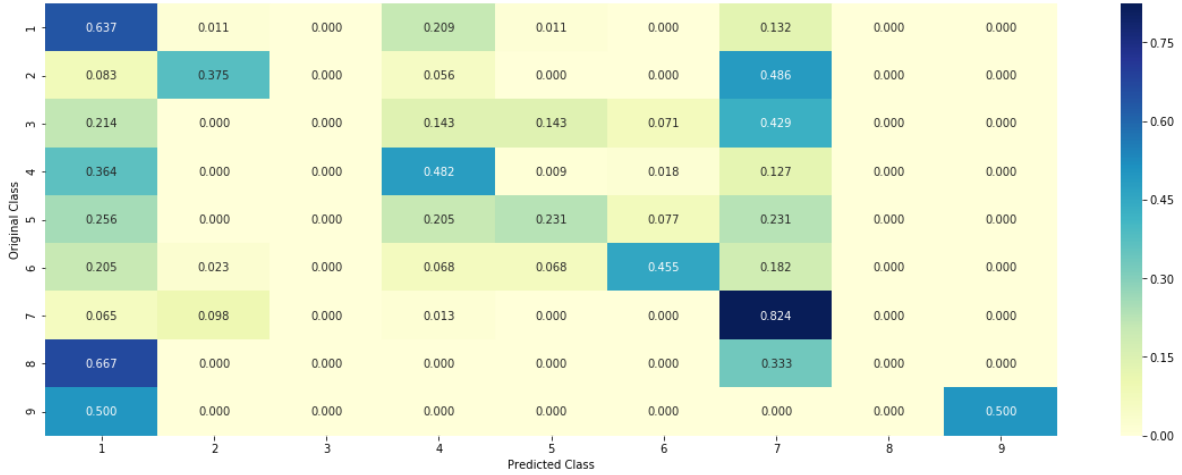
```
Log loss : 1.208019866620507
Number of mis-classified points : 0.44360902255639095
-------------------- Confusion matrix --------------------
```




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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [0]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1943 0.0309 0.0119 0.6265 0.0415 0.0
414 0.0433 0.0051 0.0051]]
Actual Class : 5
--------------------------------------------------
3 Text feature [suppressor] present in test data point [True]
4 Text feature [function] present in test data point [True]
6 Text feature [activation] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
10 Text feature [missense] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [pten] present in test data point [True]
16 Text feature [cells] present in test data point [True]
18 Text feature [loss] present in test data point [True]
20 Text feature [defective] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [functional] present in test data point [True]
30 Text feature [57] present in test data point [True]
31 Text feature [repair] present in test data point [True]
```

```
41 Text feature [expression] present in test data point [True]
43 Text feature [akt] present in test data point [True]
45 Text feature [cell] present in test data point [True]
47 Text feature [functions] present in test data point [True]
53 Text feature [activate] present in test data point [True]
55 Text feature [treated] present in test data point [True]
56 Text feature [phosphatase] present in test data point [True]
58 Text feature [predicted] present in test data point [True]
64 Text feature [inactivation] present in test data point [True]
65 Text feature [oncogene] present in test data point [True]
68 Text feature [clinical] present in test data point [True]
71 Text feature [dna] present in test data point [True]
75 Text feature [downstream] present in test data point [True]
79 Text feature [patients] present in test data point [True]
89 Text feature [damage] present in test data point [True]
92 Text feature [risk] present in test data point [True]
95 Text feature [likely] present in test data point [True]
97 Text feature [activity] present in test data point [True]
98 Text feature [null] present in test data point [True]
99 Text feature [transfected] present in test data point [True]
Out of the top  100  features  34 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [0]:

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

```
Predicted Class : 7
```

```
Predicted Class Probabilities: [[0.0289 0.1468 0.094  0.0361 0.0372 0.0
319 0.6191 0.0037 0.0023]]
Actuall Class : 3
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitors] present in test data point [True]
4 Text feature [function] present in test data point [True]
6 Text feature [activation] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
10 Text feature [missense] present in test data point [True]
11 Text feature [activated] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [pten] present in test data point [True]
16 Text feature [cells] present in test data point [True]
17 Text feature [oncogenic] present in test data point [True]
18 Text feature [loss] present in test data point [True]
23 Text feature [therapy] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [functional] present in test data point [True]
28 Text feature [trials] present in test data point [True]
41 Text feature [expression] present in test data point [True]
42 Text feature [growth] present in test data point [True]
43 Text feature [akt] present in test data point [True]
45 Text feature [cell] present in test data point [True]
46 Text feature [proteins] present in test data point [True]
50 Text feature [months] present in test data point [True]
53 Text feature [activate] present in test data point [True]
55 Text feature [treated] present in test data point [True]
56 Text feature [phosphatase] present in test data point [True]
59 Text feature [resistance] present in test data point [True]
60 Text feature [sensitivity] present in test data point [True]
62 Text feature [assays] present in test data point [True]
68 Text feature [clinical] present in test data point [True]
71 Text feature [dna] present in test data point [True]
72 Text feature [inhibition] present in test data point [True]
79 Text feature [patients] present in test data point [True]
83 Text feature [sensitive] present in test data point [True]
```

```
87 Text feature [response] present in test data point [True]
88 Text feature [survival] present in test data point [True]
90 Text feature [expressing] present in test data point [True]
95 Text feature [likely] present in test data point [True]
96 Text feature [catalytic] present in test data point [True]
97 Text feature [activity] present in test data point [True]
98 Text feature [null] present in test data point [True]
99 Text feature [transfected] present in test data point [True]
Out of the top  100  features  44 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [0]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='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.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

```
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (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 : 2.0237128938221045
for n_estimators = 10 and max depth =  3
Log Loss : 1.8852842345361882
for n_estimators = 10 and max depth =  5
Log Loss : 1.4126703480304696
for n_estimators = 10 and max depth =  10
Log Loss : 1.7233654798832878
for n_estimators = 50 and max depth =  2
Log Loss : 1.7481795635776538
```

```
for n_estimators = 50 and max depth =  3
Log Loss : 1.5549131586794667
for n_estimators = 50 and max depth =  5
Log Loss : 1.4475421987382466
for n_estimators = 50 and max depth =  10
Log Loss : 1.6671891598525506
for n_estimators = 100 and max depth =  2
Log Loss : 1.6517265292495777
for n_estimators = 100 and max depth =  3
Log Loss : 1.5859433783249657
for n_estimators = 100 and max depth =  5
Log Loss : 1.42758552467009
for n_estimators = 100 and max depth =  10
Log Loss : 1.6674232280976256
for n_estimators = 200 and max depth =  2
Log Loss : 1.7010567382242066
for n_estimators = 200 and max depth =  3
Log Loss : 1.5716666742733434
for n_estimators = 200 and max depth =  5
Log Loss : 1.4560093712085975
for n_estimators = 200 and max depth =  10
Log Loss : 1.7638136252058616
for n_estimators = 500 and max depth =  2
Log Loss : 1.7754435520040355
for n_estimators = 500 and max depth =  3
Log Loss : 1.6433837382150607
for n_estimators = 500 and max depth =  5
Log Loss : 1.4888341113319512
for n_estimators = 500 and max depth =  10
Log Loss : 1.8029338758617912
for n_estimators = 1000 and max depth =  2
Log Loss : 1.755020656304373
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6486263532873828
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4814490267726692
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7487356860615932
For values of best alpha =  10 The train log loss is: 0.069349223463484
42
```

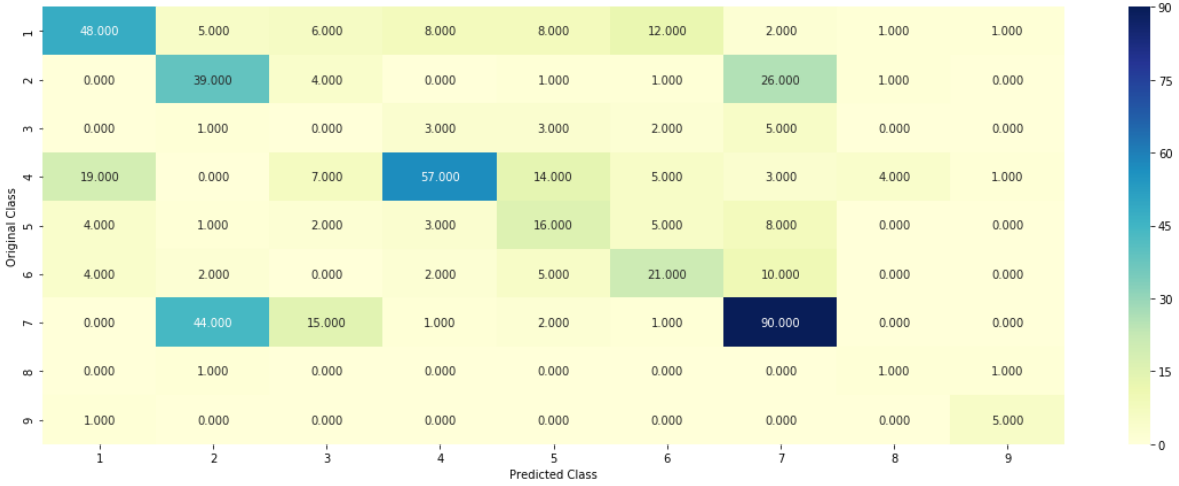
```
For values of best alpha =  10 The cross validation log loss is: 1.4126
703480304696
For values of best alpha =  10 The test log loss is: 1.4025607631142138
```

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

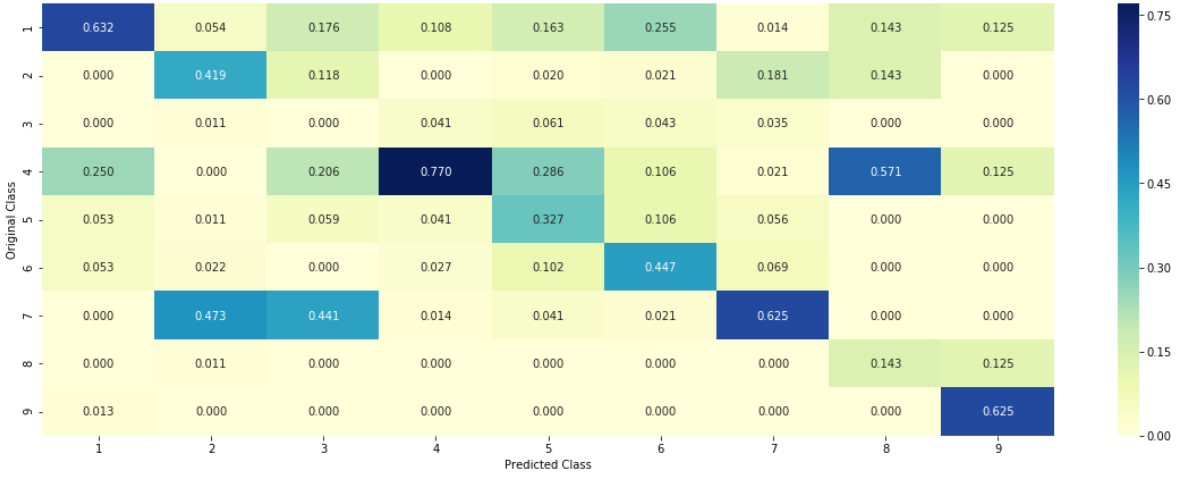
In [0]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# 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 : 1.4126703480304696
```

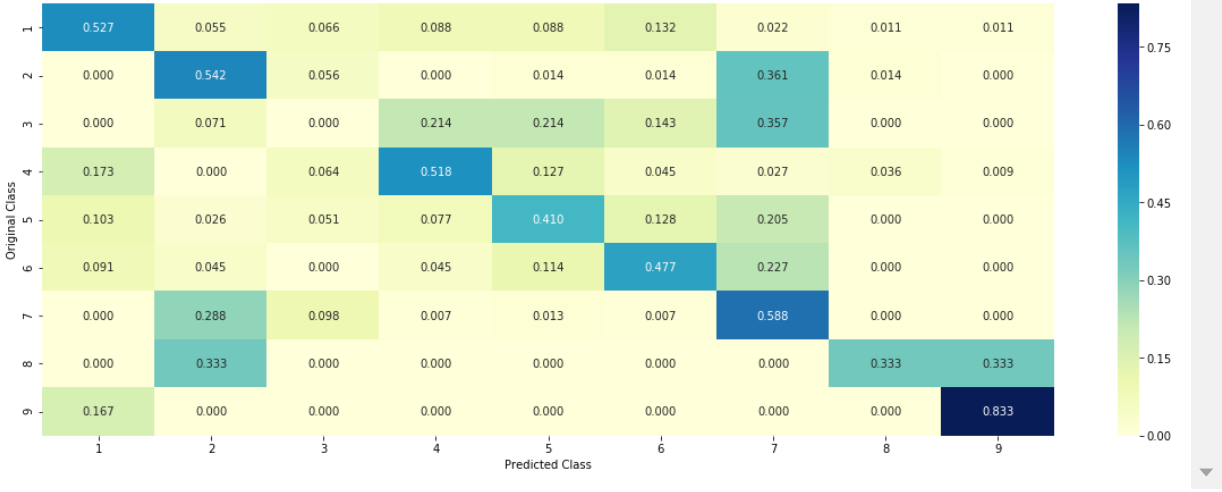
```
Number of mis-classified points : 0.4793233082706767
-------------------- Confusion matrix --------------------
```




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




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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [0]:
```
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 : 4
Predicted Class Probabilities: [[0.0485 0.0257 0.1588 0.6016 0.0116 0.0
272 0.0258 0.0601 0.0406]]
Actual Class : 5
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

```
Gene is important feature


Gene is important feature
Text is important feature
```

4.5.5.2. Incorrectly Classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0097 0.1463 0.3439 0.014  0.0463 0.0
344 0.3701 0.023  0.0125]]
Actual Class : 3
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [0]:

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

# some of methods
# fit(X, y[, coef_init, intercept_init, ...])     Fit linear model with Stochastic Gradient Descent.
```

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

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


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

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


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

```
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the 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
```

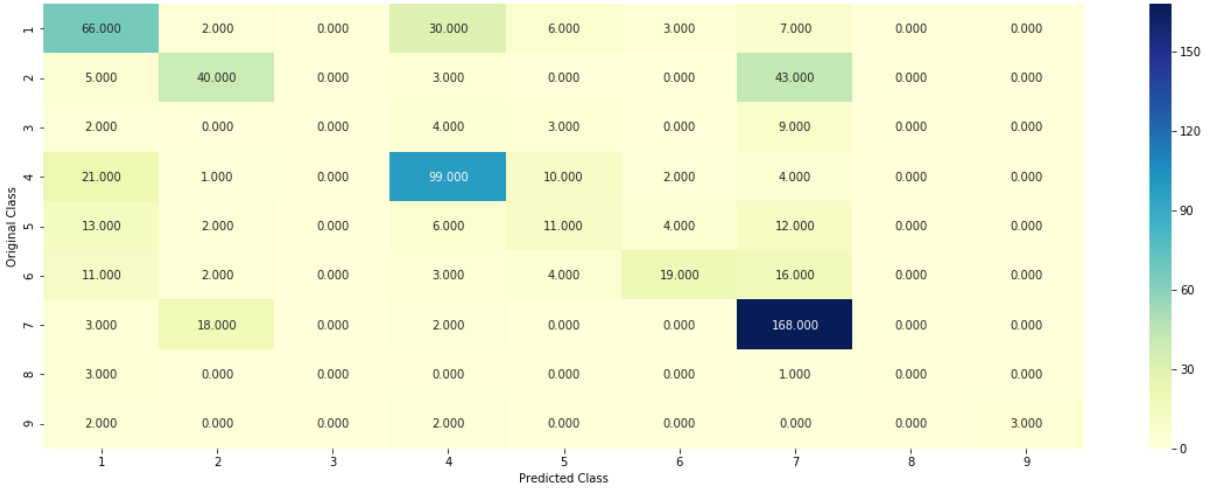
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.85
Naive Bayes : Log Loss: 1.25
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.540
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.214
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.247
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.352
```

4.7.2 testing the model with the best hyper parameters

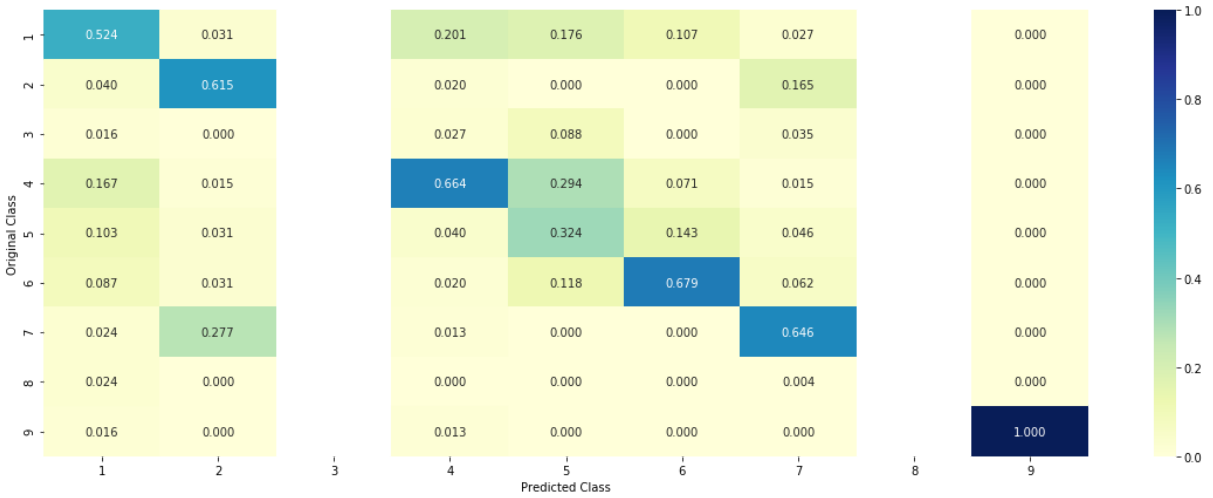
In [0]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.8064727309479287
Log loss (CV) on the stacking classifier : 1.2138379551465672
Log loss (test) on the stacking classifier : 1.1918580288635152
Number of missclassified point : 0.3894736842105263
-------------------- Confusion matrix --------------------
```




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

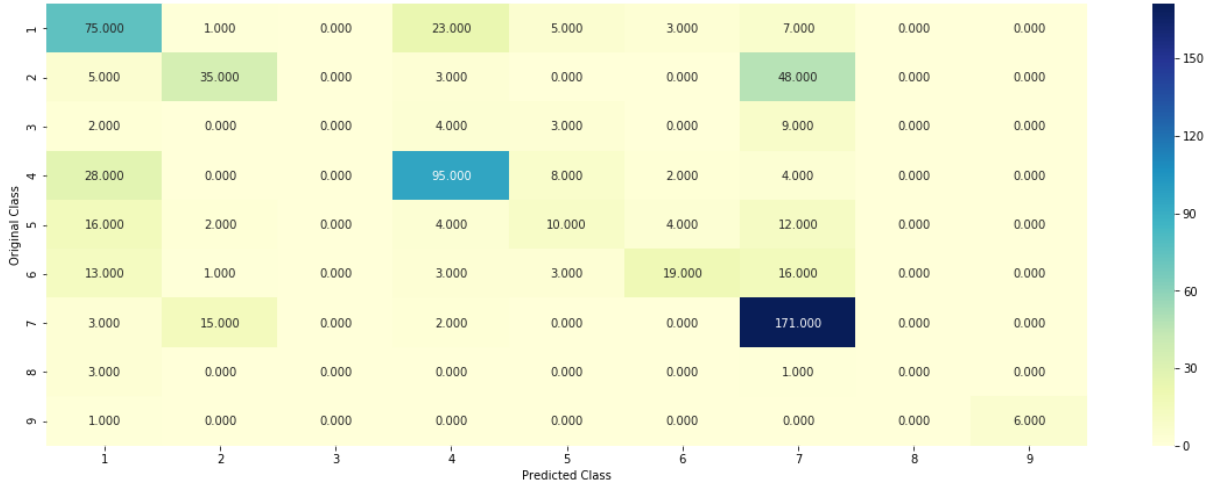


---------- Recall matrix (Row sum=1) ----------

4.7.3 Maximum Voting classifier

In [0]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
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))
```

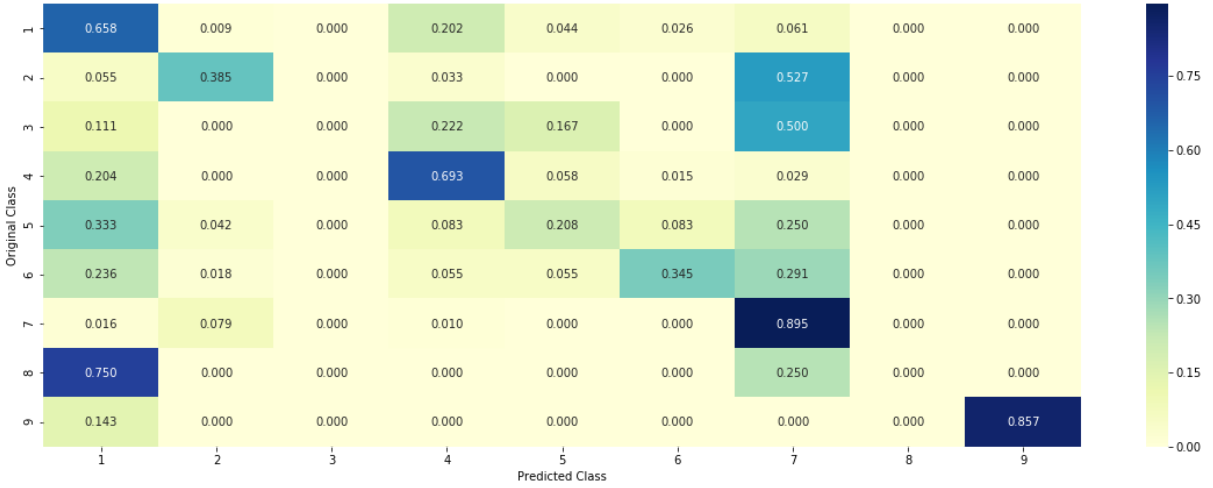
```
Log loss (train) on the VotingClassifier : 0.9360419387951573
Log loss (CV) on the VotingClassifier : 1.2375859787009489
Log loss (test) on the VotingClassifier : 1.2354250892599936
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```




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

- -

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



5. Assignments

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

3.Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [16]:
```
train_df.head()
```

Out[16]:

	ID	Gene	Variation	Class	TEXT
1642	1642	FLT3	N676K	7	8 21 inv 16 16 16 rearrangements affecting cor...
1260	1260	PIK3R1	R574fs	4	abstract cancer specific mutations ish2 inter ...

	ID	Gene	Variation	Class	TEXT
927	927	PDGFRA	ETV6-PDGFRA_Fusion	7	identified two patients 2 4 p24 q12 4 12 q2 3 ...
1896	1896	MTOR	L1460P	7	mammalian target rapamycin mtor serine threoni...
1710	1710	POLE	P286R	4	tumors somatic mutations proofreading exonucle...

In [0]:
```
#onehot encoding of gene feature
gene_vec = CountVectorizer(ngram_range=(1,2))
train_gene_onehot = gene_vec.fit_transform(train_df['Gene'])
test_gene_onehot = gene_vec.transform(test_df['Gene'])
cv_gene_onehot = gene_vec.transform(cv_df['Gene'])
```

In [0]:
```
#onehot encoding of variation feature
Variation_vec = CountVectorizer(ngram_range=(1,2))
train_Variation_onehot = Variation_vec.fit_transform(train_df['Variatio
n'])
test_Variation_onehot = Variation_vec.transform(test_df['Variation'])
cv_Variation_onehot = Variation_vec.transform(cv_df['Variation'])
```

In [0]:
```
#onehot encoding of text feature
text_vec = CountVectorizer(ngram_range=(1,2))
train_text_onehot = text_vec.fit_transform(train_df['TEXT'])
test_text_onehot = text_vec.transform(test_df['TEXT'])
cv_text_onehot = text_vec.transform(cv_df['TEXT'])
```

In [0]:
```
train_gene_var_onehot = hstack([train_gene_onehot,train_Variation_oneho
t])
test_gene_var_onehot = hstack([test_gene_onehot,test_Variation_onehot])
cv_gene_var_onehot = hstack([cv_gene_onehot,cv_Variation_onehot])

train_x_onehot = hstack([train_gene_var_onehot,train_text_onehot]).tocs
r()
```

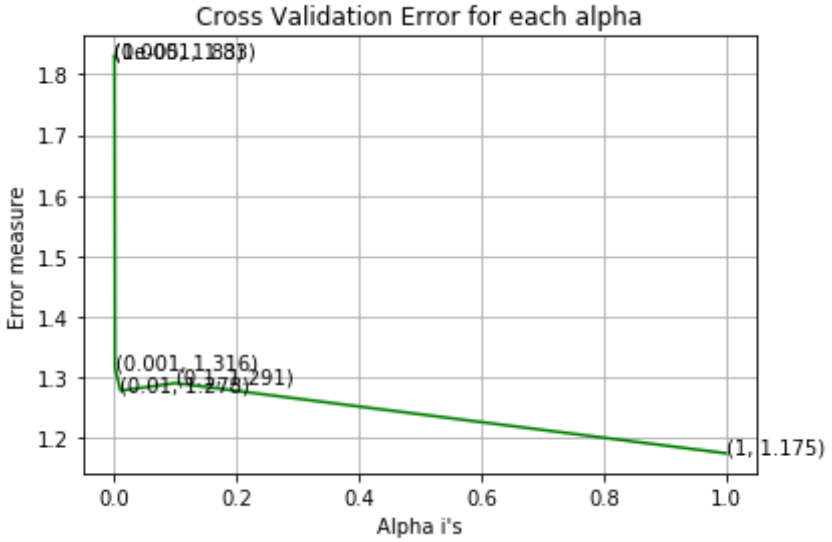
```
test_x_onehot = hstack([test_gene_var_onehot,test_text_onehot]).tocsr()
cv_x_onehot = hstack([cv_gene_var_onehot,cv_text_onehot]).tocsr()
```

In [29]:

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(class_weight='balanced',alpha=i, penalty='l2',
loss='log', random_state=42)
    clf.fit(train_x_onehot, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehot, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehot)
    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_x_onehot, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehot, y_train)
```

```
predict_y = sig_clf.predict_proba(train_x_onehot)
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_onehot)
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_onehot)
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))
```

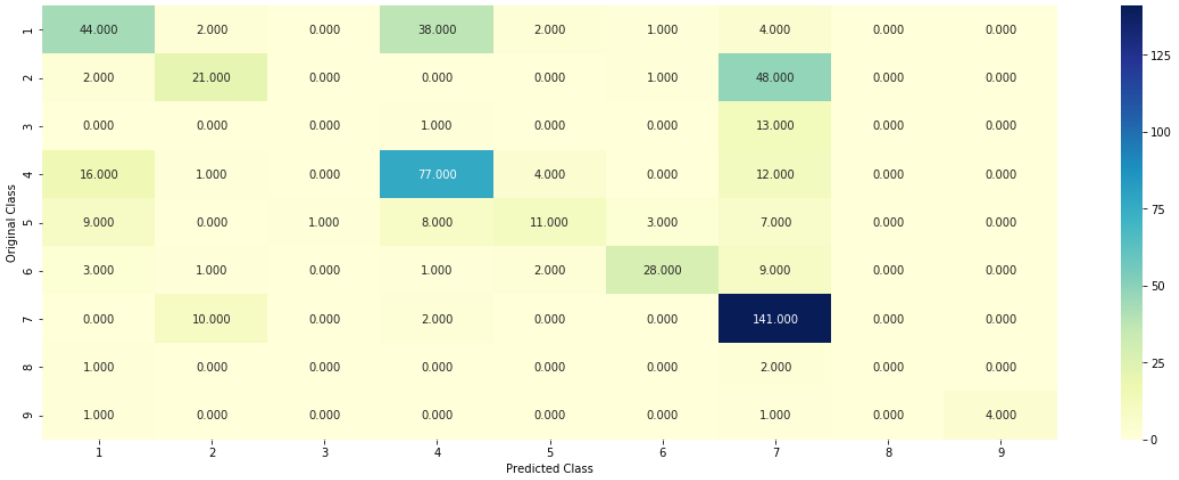
```
For values of alpha =  1e-05 The log loss is: 1.8304997567764278
For values of alpha =  0.0001 The log loss is: 1.8304997567764278
For values of alpha =  0.001 The log loss is: 1.3158235945906254
For values of alpha =  0.01 The log loss is: 1.278483331958866
For values of alpha =  0.1 The log loss is: 1.2910175139736126
For values of alpha =  1 The log loss is: 1.175206129756205
```



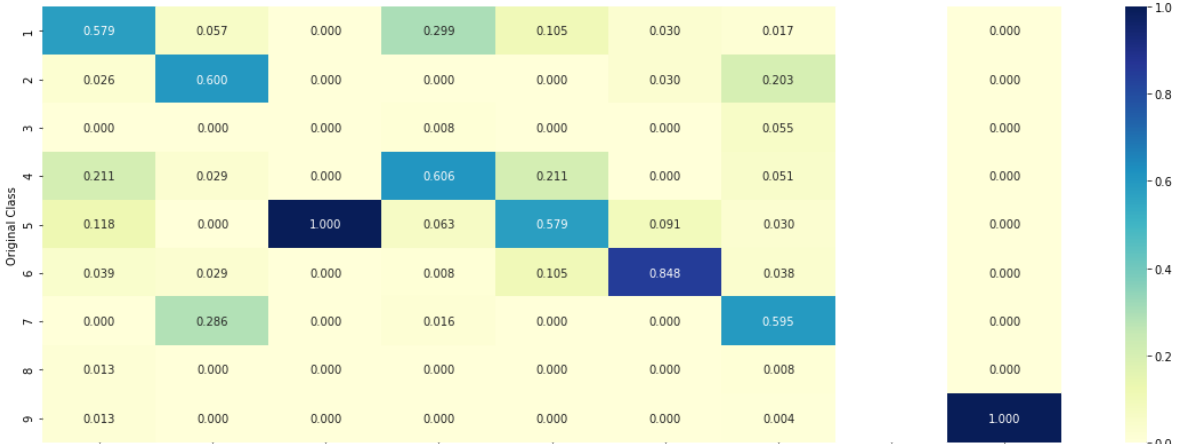

```
For values of best alpha =  1 The train log loss is: 0.8518101459242716
For values of best alpha =  1 The cross validation log loss is: 1.17545
0672863675
For values of best alpha =  1 The test log loss is: 1.2037567857558809
```

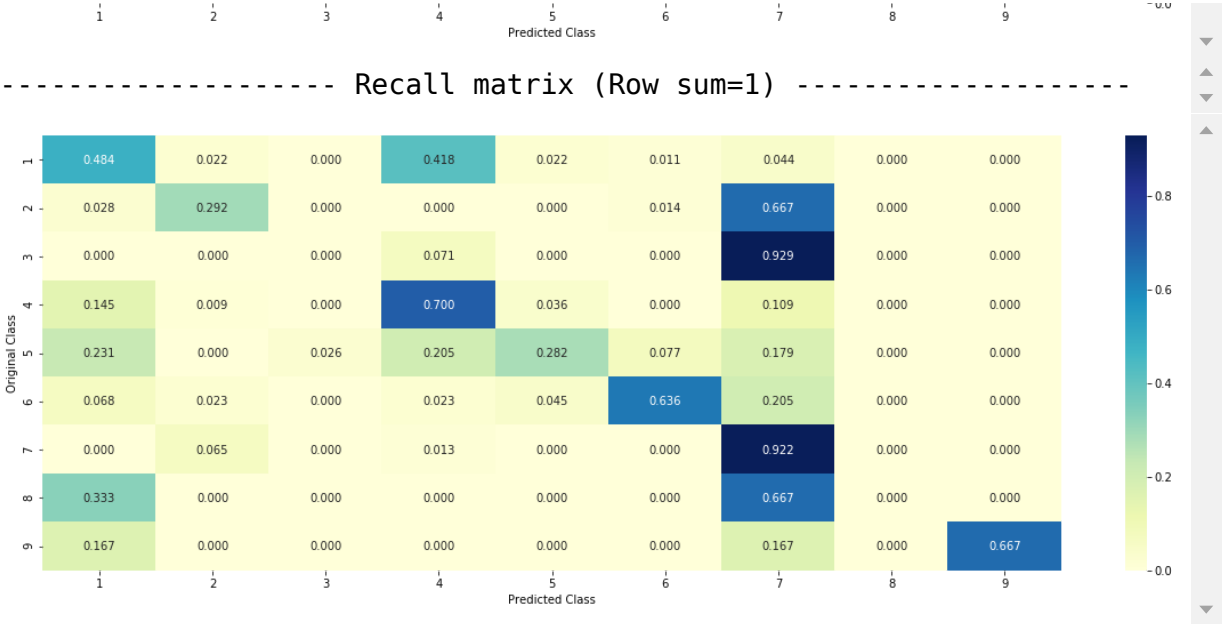
In [32]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehot, y_train, cv_x_onehot,
 y_cv, clf)
```

```
Log loss : 1.175206129756205
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```




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





4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

In [0]:
```
#onehot encoding of gene feature
gene_vec = TfidfVectorizer(ngram_range=(1,2))
train_gene_onehot = gene_vec.fit_transform(train_df['Gene'])
test_gene_onehot = gene_vec.transform(test_df['Gene'])
cv_gene_onehot = gene_vec.transform(cv_df['Gene'])
```

In [0]:
```
#onehot encoding of variation feature
Variation_vec = TfidfVectorizer(ngram_range=(1,2))
train_Variation_onehot = Variation_vec.fit_transform(train_df['Variatio
n'])
test_Variation_onehot = Variation_vec.transform(test_df['Variation'])
cv_Variation_onehot = Variation_vec.transform(cv_df['Variation'])
```

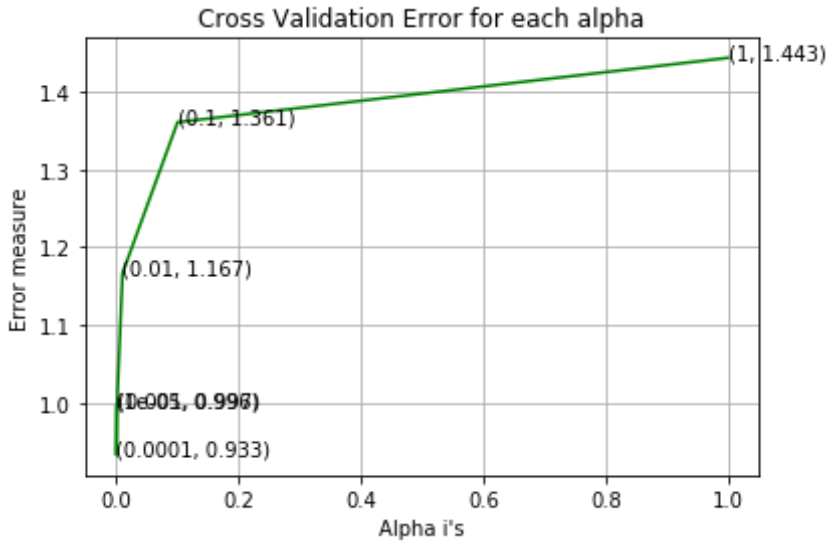
In [0]:
```
#onehot encoding of text feature
text_vec = TfidfVectorizer(ngram_range=(1,2))
train_text_onehot = text_vec.fit_transform(train_df['TEXT'])
test_text_onehot = text_vec.transform(test_df['TEXT'])
cv_text_onehot = text_vec.transform(cv_df['TEXT'])
```

In [0]:
```
train_gene_var_onehot = hstack([train_gene_onehot,train_Variation_oneho
t])
test_gene_var_onehot = hstack([test_gene_onehot,test_Variation_onehot])
cv_gene_var_onehot = hstack([cv_gene_onehot,cv_Variation_onehot])

train_x_onehot = hstack([train_gene_var_onehot,train_text_onehot]).tocs
r()
test_x_onehot = hstack([test_gene_var_onehot,test_text_onehot]).tocsr()
cv_x_onehot = hstack([cv_gene_var_onehot,cv_text_onehot]).tocsr()
```

In [42]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifie
r.


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(class_weight='balanced',alpha=i, penalty='l2',
loss='log', random_state=42)
    clf.fit(train_x_onehot, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehot, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehot)
    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_x_onehot, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehot, y_train)

predict_y = sig_clf.predict_proba(train_x_onehot)
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_onehot)
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_onehot)
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))
```

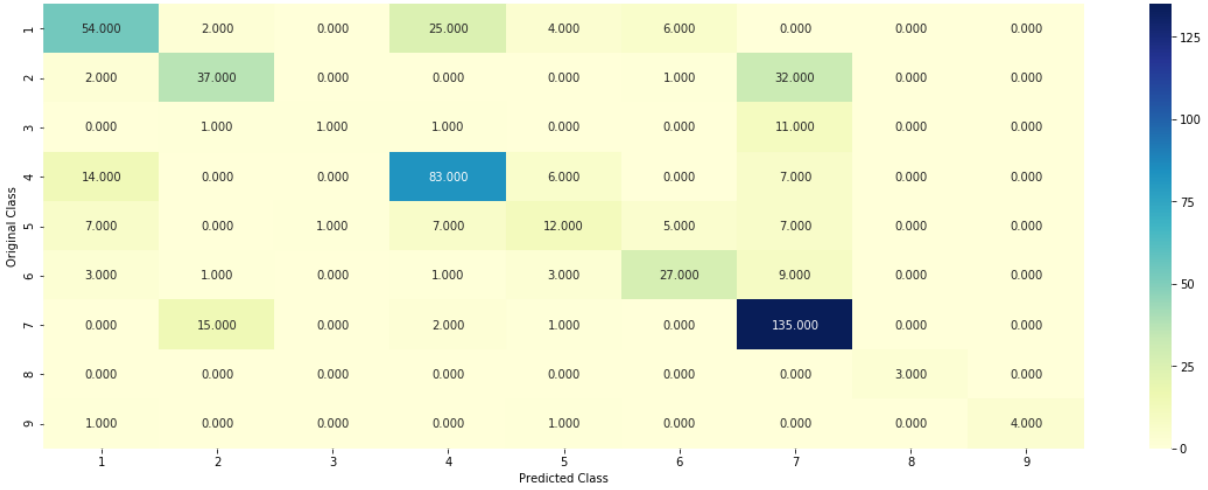
```
For values of alpha =  1e-05 The log loss is: 0.9962259307593575
For values of alpha =  0.0001 The log loss is: 0.9328356278570592
For values of alpha =  0.001 The log loss is: 0.9965975340476788
For values of alpha =  0.01 The log loss is: 1.166715072325909
For values of alpha =  0.1 The log loss is: 1.3606204858200783
For values of alpha =  1 The log loss is: 1.4433209841200594
```




```
For values of best alpha =  0.0001 The train log loss is: 0.40833820113
03158
For values of best alpha =  0.0001 The cross validation log loss is: 0.
929123113244849
For values of best alpha =  0.0001 The test log loss is: 0.982035085438
0745
```

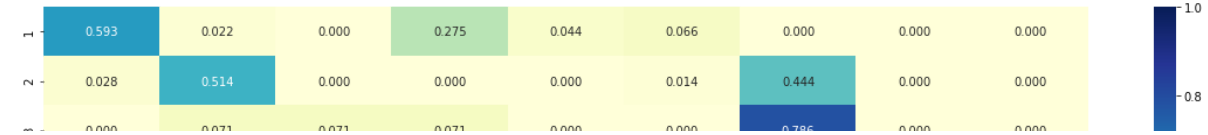
In [43]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehot, y_train, cv_x_onehot,
 y_cv, clf)
```

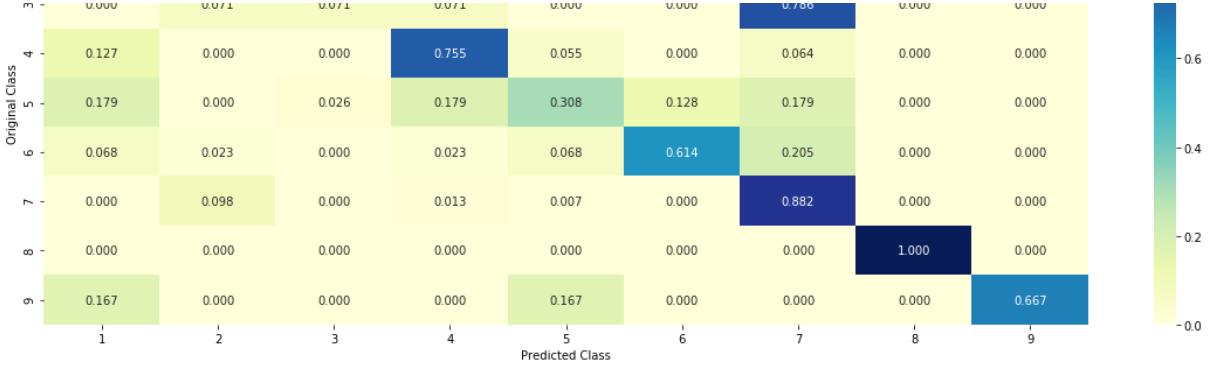
```
Log loss : 0.9328356278570592
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```



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

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





In [48]:

```
df = pd.DataFrame({"Model":["Naive Bayes(onehot)","KNN","Logistic Regression(with balancing)(onehot)","Logistic Regression(without balancing)(onehot)","Linear SVM(with balancing)(onehot)","Random Forest Classifier(onehot)","Random Forest Classifier(response coding)","Stacking (NB,SVM,LR)(onehot)","Maximum Voting classifier(LR,SVM,RF)(onehot)","Logistic regression(unigrams and bigrams)","Logistic regression(after feature engoneering)"],
                   "Train log loss":[0.7394963604785211,0.4772188583588
3404,0.5558921845210903,0.5534125966965848,0.7893184207314842,0.8432100
558401556,0.06934922346348442,0.8064727309479287,0.9360419387951573,0.8
518101459242716,0.4083382011303158],
                                     "cv log loss":[1.25459709762871
1,1.1039685507401267,1.0668751749577636,1.0847711857361417,1.1576852813
95595,1.2080198666205069,1.4126703480304696,1.2138379551465672,1.237585
9787009489,1.175450672863675,0.929123113244849],
                                     "Test log loss":[1.237384254148
2603,1.1383043424426245,1.0595860348918453,1.0807995586747963,1.1888444
```

```
001752025,1.2544436898616733,1.4025607631142138,1.1918580288635152,1.23
54250892599936,1.2037567857558809,0.9820350854380745],
                                                    "% of miss classified points":[
42.85,38.34,34.58,35.90,35.90,44.36,47.93,38.94,38.19,38.72,33.08]
}
                        ,columns=["Model","Train log loss","cv log loss","Tes
t log loss","% of miss classified points"])
df.sort_values(by="% of miss classified points",ascending=True)
```

Out[48]:

	Model	Train log loss	cv log loss	Test log loss	% of miss classified points
10	Logistic regression(after feature engoneerong)	0.408338	0.929123	0.982035	33.08
2	Logistic Regression(with balancing)(onehot)	0.555892	1.066875	1.059586	34.58
3	Logistic Regression(without balancing)(onehot)	0.553413	1.084771	1.080800	35.90
4	Linear SVM(with balancing)(onehot)	0.789318	1.157685	1.188844	35.90
8	Maximum Voting classifier(LR,SVM,RF)(onehot)	0.936042	1.237586	1.235425	38.19
1	KNN	0.477219	1.103969	1.138304	38.34
9	Logistic regression(unigrams and bigrams)	0.851810	1.175451	1.203757	38.72
7	Stacking (NB,SVM,LR)(onehot)	0.806473	1.213838	1.191858	38.94
0	Naive Bayes(onehot)	0.739496	1.254597	1.237384	42.85
5	Random Forest Classifier(onehot)	0.843210	1.208020	1.254444	44.36
6	Random Forest Classifier(response coding)	0.069349	1.412670	1.402561	47.93

Here above result shows that after feature engineering cv log loss is less than 1 in logistic regression