PersonalizedCancerDiagnosis

March 20, 2019

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from 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
```

```
In [2]: import os.path
        os.path.isfile('training/training_text.csv')
```

```
Out[2]: True
```

```
In [3]: data=pd.read_csv('training/training_variants.csv')
        print('number of data points={}'.format(data.shape[0]))
        data.head()
```

```
number of data points=3321
```

```
Out[3]:    ID    Gene             Variation  Class
        0   0  FAM58A  Truncating Mutations      1
        1   1     CBL                 W802*      2
        2   2     CBL                 Q249E      2
        3   3     CBL                 N454D      3
        4   4     CBL                 L399V      4
```

```
In [4]: data_text=pd.read_csv('training/training_text.csv',sep="\|\|",engine="python",names=["
        print('number of data points={}'.format(data_text.shape[0]))
        data.head()
```

```
number of data points=3321
```

```
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
```

```
In [12]: 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 word not in stop_words:
                         string+=word+" "
                 data_text[column][index]=string
```

In [13]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 169.0253423005418 seconds
```

In [14]:
```
result=pd.merge(data,data_text,on='ID',how='left')
result.head()
```

Out[14]:

	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	

	TEXT
0	cyclin dependent kinases cdks regulate variety...
1	abstract background non small cell lung cancer...
2	abstract background non small cell lung cancer...
3	recent evidence demonstrated acquired uniparen...
4	oncogenic mutations monomeric casitas b lineag...

In [15]:
```
result[result.isnull().any(axis=1)]
```

Out[15]:

	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 [16]:
```
result.loc[result['TEXT'].isnull(),'TEXT']=result['Gene']+" "+result['Variation']
```

In [17]:
```
result[result['ID']==1109]
```

```
Out[17]:         ID   Gene Variation  Class          TEXT
         1109  1109  FANCA    S1088F      1  FANCA S1088F
```

```
In [18]: y_true=result['Class'].values
         result.Gene=result.Gene.str.replace('\s+',' ')
         result.Variation=result.Variation.str.replace('\s+',' ')
         # split the data into test and train by maintaining same distribution of output varai
         X_train,test_df,y_train,y_test=train_test_split(result,y_true,stratify=y_true,test_siz

         # split the train data into train and cross validation by maintaining same distributi
         train_df,cv_df,y_train,y_cv=train_test_split(X_train,y_train,stratify=y_train,test_siz
```

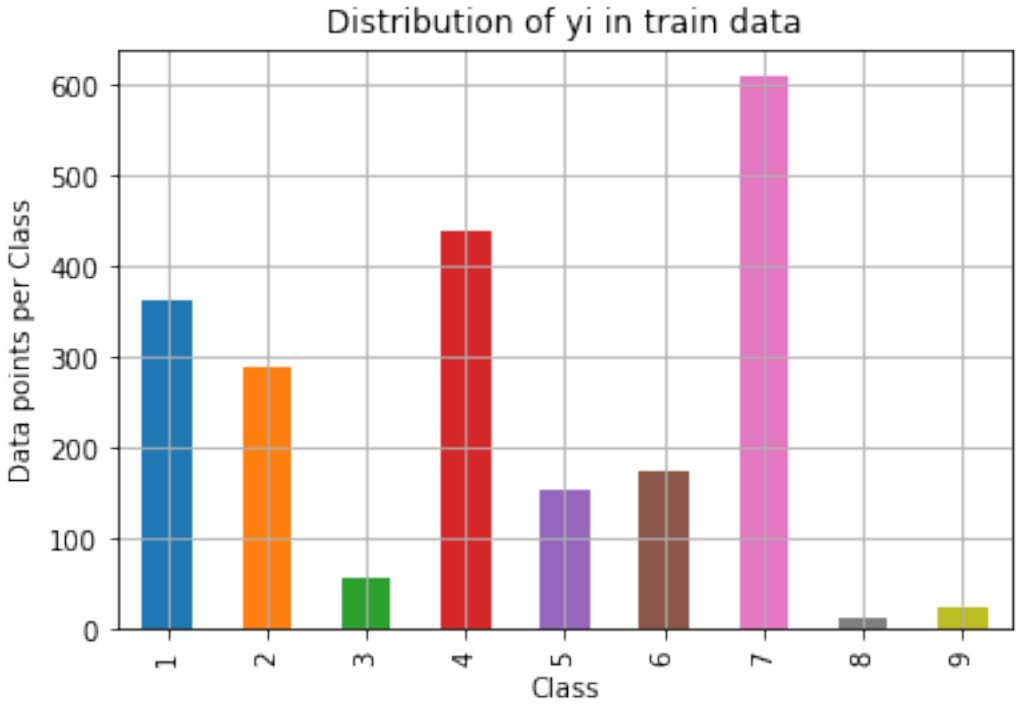
```
In [19]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

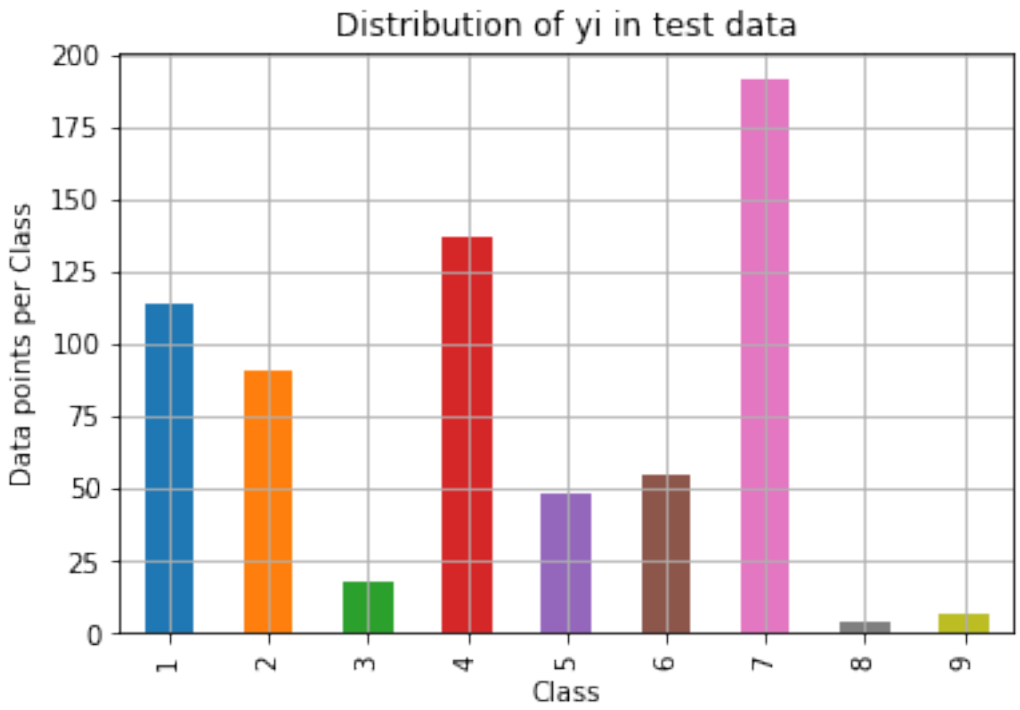
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [20]: train_class_distribution=train_df['Class'].value_counts().sort_index()
         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()
         for index,value in dict(train_df['Class'].value_counts().sort_index()).items():
             print('the number of data points in class {} is {},{:0.2f}%'.format(index,value,(v
         print('-'*80)
         test_class_distribution=test_df['Class'].value_counts().sort_index()
         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()
         for index,value in dict(test_df['Class'].value_counts().sort_index()).items():
             print('the number of data points in class {} is {},{:0.2f}%'.format(index,value,(v
         print('-'*80)
         cv_class_distribution=cv_df['Class'].value_counts().sort_index()
         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()
```

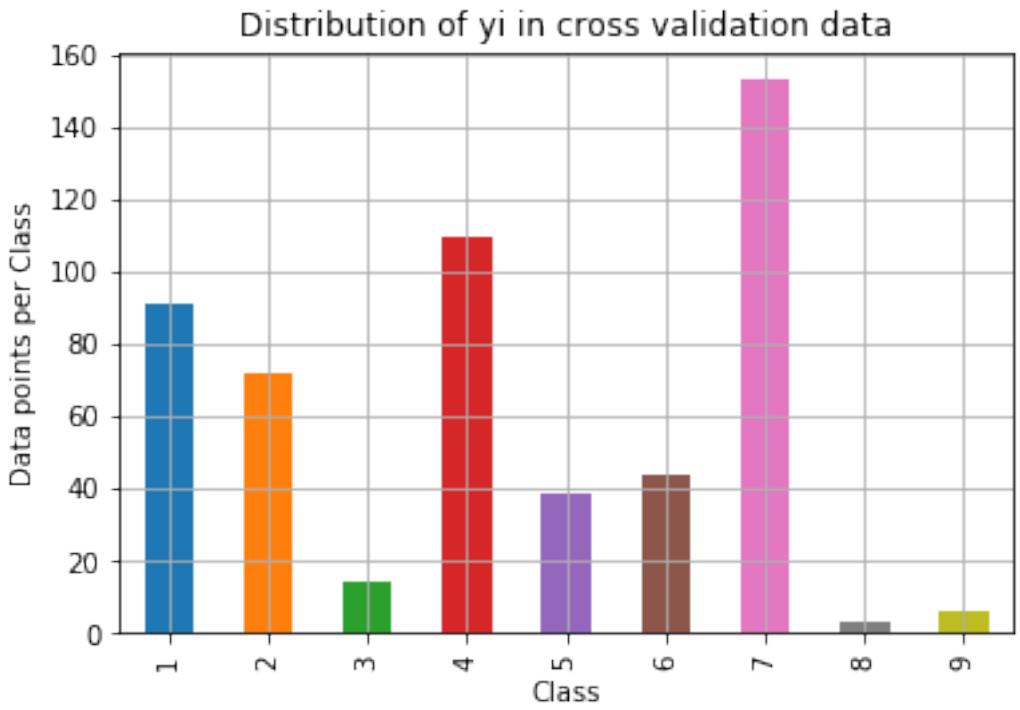
```
plt.show()
for index,value in dict(cv_df['Class'].value_counts().sort_index()).items():
    print('the number of data points in class {} is {},{:0.2f}%'.format(index,value,(
print('-'*80)
```




```
the number of data points in class 1 is 363,17.09%
the number of data points in class 2 is 289,13.61%
the number of data points in class 3 is 57,2.68%
the number of data points in class 4 is 439,20.67%
the number of data points in class 5 is 155,7.30%
the number of data points in class 6 is 176,8.29%
the number of data points in class 7 is 609,28.67%
the number of data points in class 8 is 12,0.56%
the number of data points in class 9 is 24,1.13%
--------------------------------------------------------------------------------
```




```
the number of data points in class 1 is 114,5.37%
the number of data points in class 2 is 91,4.28%
the number of data points in class 3 is 18,0.85%
the number of data points in class 4 is 137,6.45%
the number of data points in class 5 is 48,2.26%
the number of data points in class 6 is 55,2.59%
the number of data points in class 7 is 191,8.99%
the number of data points in class 8 is 4,0.19%
the number of data points in class 9 is 7,0.33%
------------------------------------------------------------------------
```




```
the number of data points in class 1 is 91,4.28%
the number of data points in class 2 is 72,3.39%
the number of data points in class 3 is 14,0.66%
the number of data points in class 4 is 110,5.18%
the number of data points in class 5 is 39,1.84%
the number of data points in class 6 is 44,2.07%
the number of data points in class 7 is 153,7.20%
the number of data points in class 8 is 3,0.14%
the number of data points in class 9 is 6,0.28%
--------------------------------------------------------------------------------
```

3.2 Prediction using a 'Random' Model

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

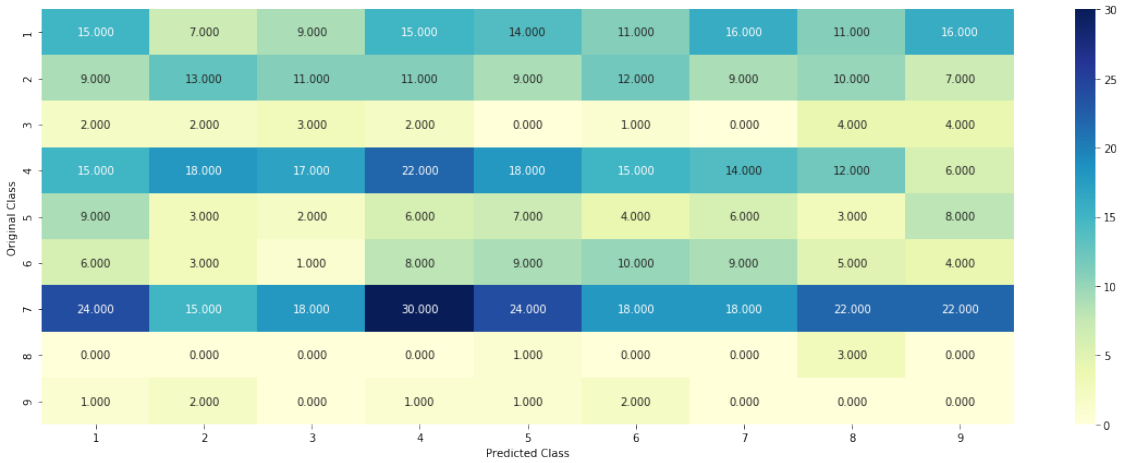
```
In [21]: # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
             # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

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

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

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

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

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

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

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

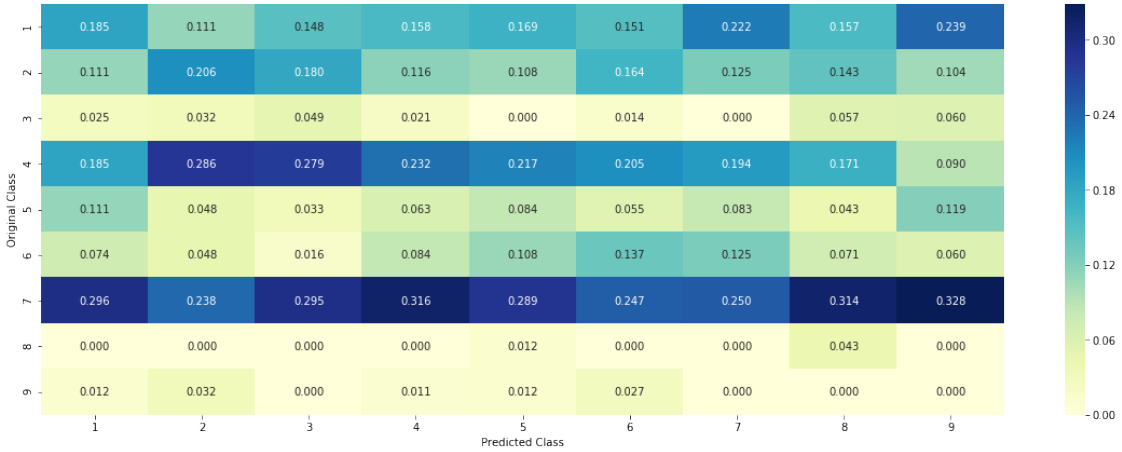
In [22]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
```

```
# ref: https://stackoverflow.com/a/18662466/4084039
cv_data_len=cv_df.shape[0]
test_data_len=test_df.shape[0]
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))
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicte


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))
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y,eps=

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

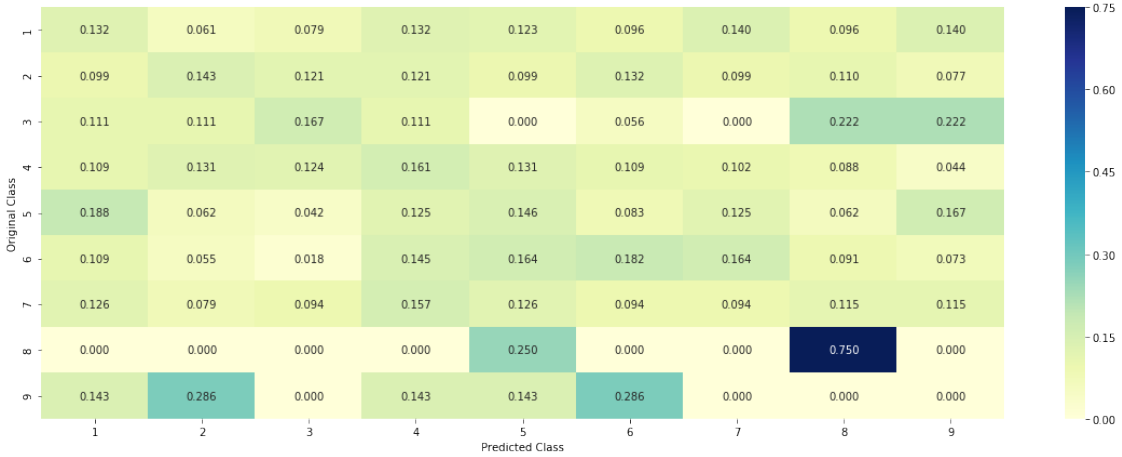
```
Log loss on Cross Validation Data using Random Model 2.4951000886147265
Log loss on Test Data using Random Model 2.3832272093705886
-------------------- Confusion matrix --------------------
```




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



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



3.3 Univariate Analysis

```
In [23]: value_count=train_df['Variation'].value_counts()
         value_count.head(2)

Out[23]: Truncating Mutations    60
         Deletion                47
         Name: Variation, dtype: int64

In [24]: c=0
         for index, row in train_df.iterrows():
             print(index,row)
             print(row['Variation'])
```

```
            c+=1
            if c==2:
                break
```

```
2426 ID                                                        2426
Gene                                                         BRCA1
Variation                                                     C64G
Class                                                            4
TEXT         published analyses effects missense mutations ...
Name: 2426, dtype: object
C64G
636 ID                                                         636
Gene                                                       CDKN1A
Variation                                  Truncating Mutations
Class                                                            1
TEXT         introduction loss control mammalian cell cycle...
Name: 636, dtype: object
Truncating Mutations
```

In [25]:
```
# get_gv_fea_dict: Get Gene varaition Feature Dict


def get_gv_fea_dict(alpha, feature, df):

    value_count = train_df[feature].value_counts()
    gv_dict={}
    #fea=[num of times it occ in class 1+10 alpha/num of times it occured in totla da
    for i,value in value_count.items():
        vec=[]
        for k in range(1,10):
            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
            b=(sum(cls_cnt['Class'])+10*alpha)/(value+90*alpha)
            vec.append(b)
        gv_dict[i]=vec
    return gv_dict

def get_gv_feature(alpha, feature, df):
    gv_fea=[]
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    for index, row in df.iterrows():
        if row[feature] in gv_dict.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])
    return gv_fea
```

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?

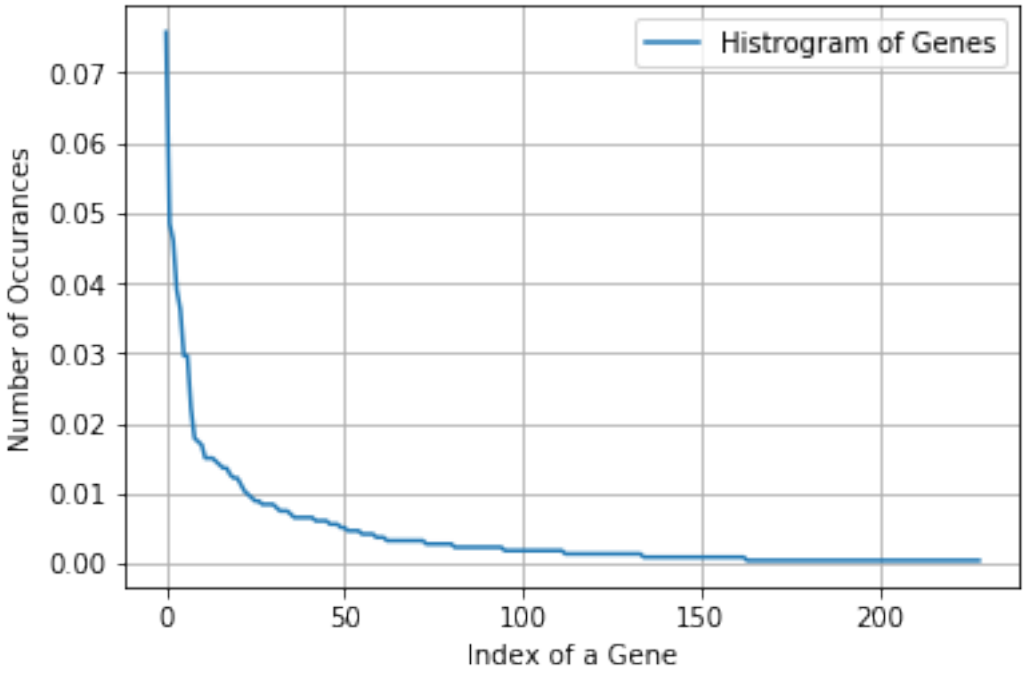
```
In [26]: unique_genes = train_df['Gene'].value_counts()
         print('Number of Unique Genes :', unique_genes.shape[0])
         # the top 10 genes that occured most
         print(unique_genes.head(10))
```

```
Number of Unique Genes : 229
BRCA1     161
TP53      103
EGFR       98
PTEN       83
BRCA2      77
BRAF       63
KIT        63
ERBB2      47
ALK        38
PDGFRA     37
Name: Gene, dtype: int64
```

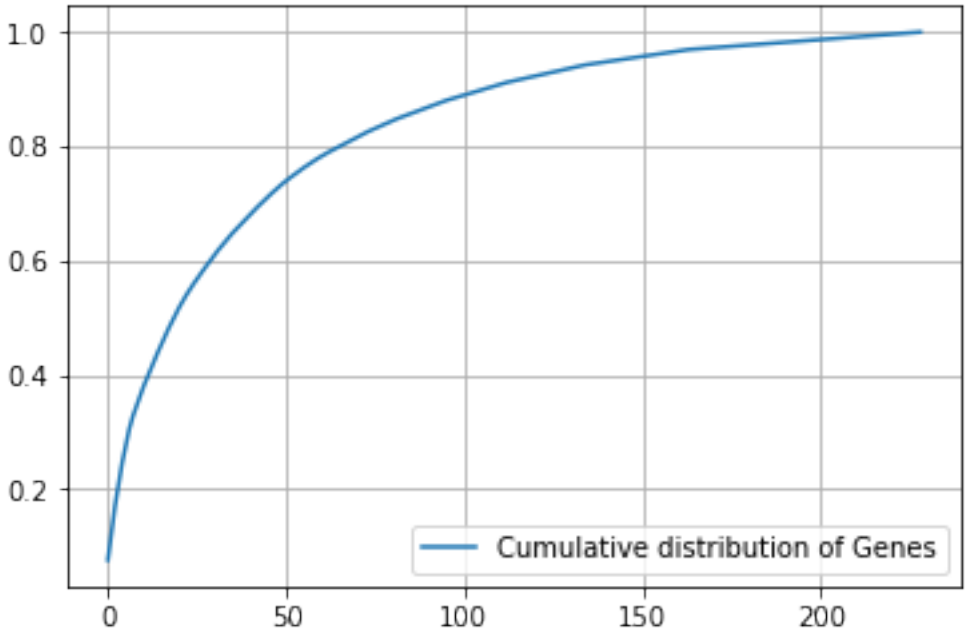
```
In [27]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the t
```

```
Ans: There are 229 different categories of genes in the train data, and they are distibuted as
```

```
In [28]: 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 [29]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



In [30]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [31]:
```
train_gene_feature_responseCoding[0]
```

Out[31]:
```
array([0.17131474, 0.03984064, 0.15936255, 0.51792829, 1.09561753,
       0.87649402, 0.03984064, 0.03984064, 0.03984064])
```

In [32]:
```
train_df.head()
```

Out[32]:

	ID	Gene	Variation	Class	\
2426	2426	BRCA1	C64G	4	
636	636	CDKN1A	Truncating Mutations	1	
703	703	ERBB2	E812K	6	
1455	1455	FGFR2	K659N	7	
103	103	MSH6	R976H	1	

	TEXT
2426	published analyses effects missense mutations ...
636	introduction loss control mammalian cell cycle...
703	purpose mutations associated resistance kinase...
1455	activating mutations tyrosine kinase domain re...
103	msh6 gene one mismatch repair genes involved h...

In [33]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding met
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape
```

In [34]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

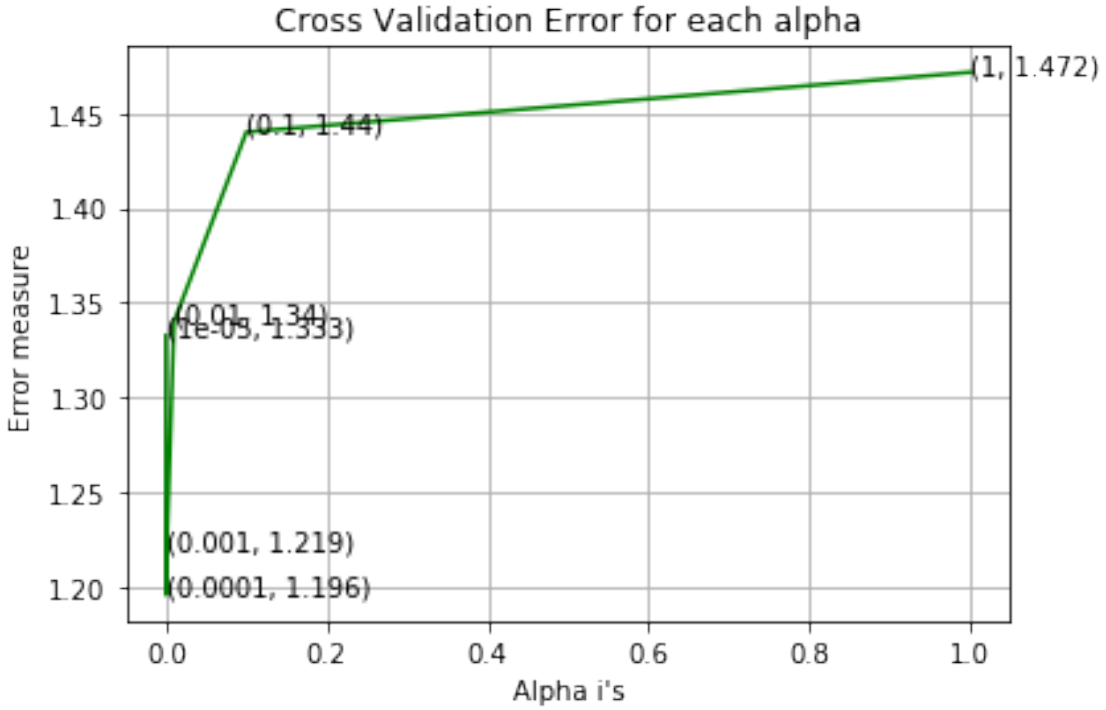
In [35]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding met
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape
```

In [36]:
```
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/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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


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

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
```

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

```
For values of alpha =  1e-05 The log loss is: 1.3326465387283037
For values of alpha =  0.0001 The log loss is: 1.1958649441508244
For values of alpha =  0.001 The log loss is: 1.219375810413702
For values of alpha =  0.01 The log loss is: 1.3397980547979844
For values of alpha =  0.1 The log loss is: 1.4400266950210705
For values of alpha =  1 The log loss is: 1.4715588366786518
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0701141579668674
For values of best alpha =  0.0001 The cross validation log loss is: 1.1958649441508244
For values of best alpha =  0.0001 The test log loss is: 1.2009072752281713
```

```
In [37]: print("Q6. How many data points in Test and CV datasets are covered by the ", unique_g

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

         print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_cover
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cov
```

```
Q6. How many data points in Test and CV datasets are covered by the  229  genes in train datase
Ans
1. In test data 639 out of 665 : 96.09022556390977
2. In cross validation data 517 out of  532 : 97.18045112781954
```

3.2.2 Univariate Analysis on Variation Feature

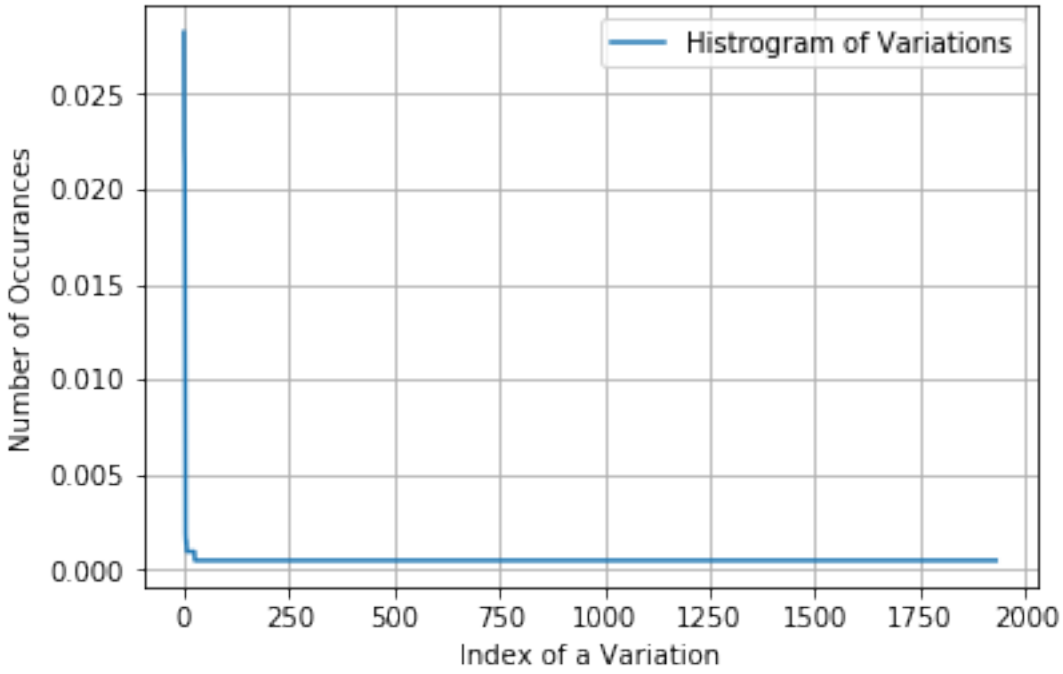
```
In [38]: unique_variations=train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         # the top 10 variations that occured most
         print(unique_variations.head(10))
```

```
Number of Unique Variations : 1931
Truncating Mutations    60
Deletion                47
Amplification           45
Fusions                 19
Overexpression           4
E17K                     3
T58I                     3
I31M                     2
A146V                    2
R170W                    2
Name: Variation, dtype: int64
```

```
In [39]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variatio
```

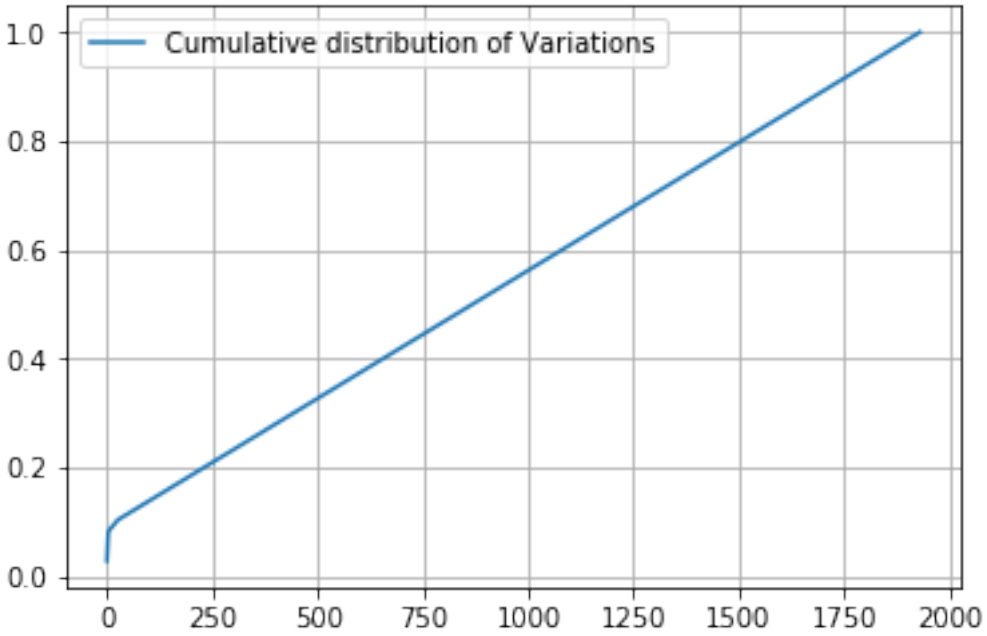
```
Ans: There are 1931 different categories of variations in the train data, and they are distibu
```

```
In [40]: 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 [41]: c = np.cumsum(h)
         print(c)
         plt.plot(c,label='Cumulative distribution of Variations')
         plt.grid()
         plt.legend()
         plt.show()
```

```
[0.02824859 0.05037665 0.07156309 ... 0.99905838 0.99952919 1.        ]
```



In [42]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_c
```

In [43]:
```
print("train_variation_feature_responseCoding is a converted feature using the respons
```

```
train_variation_feature_responseCoding is a converted feature using the response coding method
```

In [44]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [45]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot e
```

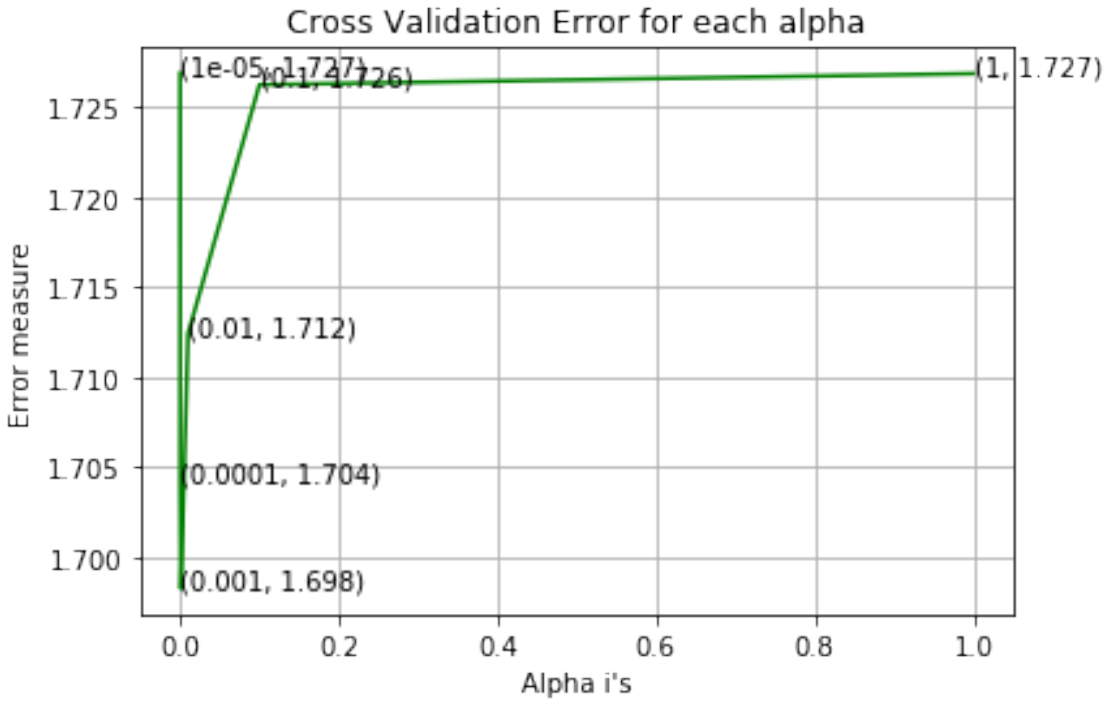
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
```

In [46]:
```
alpha = [10 ** x for x in range(-5, 1)]
```

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

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


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
```

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

```
For values of alpha =  1e-05 The log loss is: 1.7268796408083058
For values of alpha =  0.0001 The log loss is: 1.704262284350921
For values of alpha =  0.001 The log loss is: 1.6982807947202923
For values of alpha =  0.01 The log loss is: 1.7124187222286233
For values of alpha =  0.1 The log loss is: 1.7262328133557006
For values of alpha =  1 The log loss is: 1.7268511190838507
```




```
For values of best alpha =  0.001 The train log loss is: 1.1838981351925417
For values of best alpha =  0.001 The cross validation log loss is: 1.6982807947202923
For values of best alpha =  0.001 The test log loss is: 1.7113324426082255
```

```
In [47]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], "
         test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].sha
         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_cover
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_co
```

```
Q12. How many data points are covered by total  1931  genes in test and cross validation data s
Ans
```

```
1. In test data 73 out of 665 : 10.977443609022556
2. In cross validation data 54 out of  532 : 10.150375939849624
```

3.2.3 Univariate Analysis on Text Feature

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

```
In [48]: train_df.head()
```

```
Out[48]:         ID    Gene             Variation  Class  \
         2426  2426   BRCA1                  C64G      4
         636    636  CDKN1A  Truncating Mutations      1
         703    703   ERBB2                 E812K      6
         1455  1455   FGFR2                 K659N      7
         103    103    MSH6                 R976H      1

                                                      TEXT
         2426  published analyses effects missense mutations ...
         636   introduction loss control mammalian cell cycle...
         703   purpose mutations associated resistance kinase...
         1455  activating mutations tyrosine kinase domain re...
         103   msh6 gene one mismatch repair genes involved h...
```

```
In [49]: # 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 [ ]:
```

```
In [50]: import math
         #https://stackoverflow.com/a/1602964
         def get_text_responsecoding(df):
             text_feature_responseCoding = np.zeros((df.shape[0],9))
             for i in range(0,9):
                 row_index = 0
```

```
                for index, row in df.iterrows():
                    sum_prob = 0
                    for word in row['TEXT'].split():
                        sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(w
                    text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TE
                    row_index += 1
            return text_feature_responseCoding
```

In [51]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*nu
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 54022
```

In [52]:
```
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)
```

In [53]:
```
train_text_feature_onehotCoding
```

Out[53]:
```
<2124x54022 sparse matrix of type '<class 'numpy.int64'>'
        with 3353735 stored elements in Compressed Sparse Row format>
```

In [54]:
```
#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 [55]:
```
# 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_fe
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res
```

In [56]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [57]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

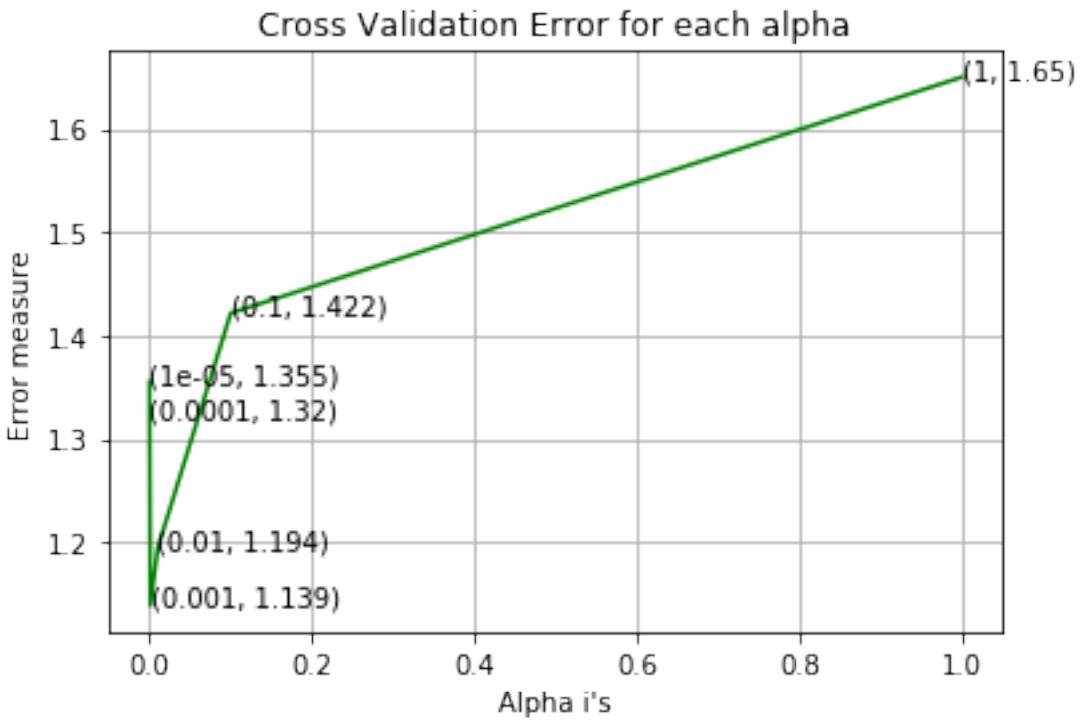
In [58]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 5582, 4: 4043, 5: 2918, 6: 2600, 9: 2213, 7: 2000, 8: 1786, 10: 1485, 12: 1334, 13
```

In [59]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot
alpha = [10 ** x for x in range(-5, 1)]

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

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


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

```
        clf.fit(train_text_feature_onehotCoding, y_train)

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

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


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

    predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
    predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.355104568056907
For values of alpha =  0.0001 The log loss is: 1.3201055961507802
For values of alpha =  0.001 The log loss is: 1.1385554981158474
For values of alpha =  0.01 The log loss is: 1.1940939749655382
For values of alpha =  0.1 The log loss is: 1.4216872588254803
For values of alpha =  1 The log loss is: 1.6501973935987877
```




```
For values of best alpha =  0.001 The train log loss is: 0.7715700661566856
For values of best alpha =  0.001 The cross validation log loss is: 1.1385554981158474
For values of best alpha =  0.001 The test log loss is: 1.1563337126549522
```

```
In [60]: def get_intersec_text(df):
             df_text_vec = CountVectorizer(min_df=3)
             df_text_fea = df_text_vec.fit_transform(df['TEXT'])
             df_text_features = df_text_vec.get_feature_names()

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

In [61]: len1,len2 = get_intersec_text(test_df)
         print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
         len1,len2 = get_intersec_text(cv_df)
         print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train d
```

```
96.851 % of word of test data appeared in train data
98.339 % of word of Cross Validation appeared in train data
```

In [62]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to
    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))/tes
    plot_confusion_matrix(test_y, pred_y)
```

In [63]:
```
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 [64]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    gene_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(w
        elif (v < fea1_len+fea2_len):
```

```
                    word = var_vec.get_feature_names()[v-(fea1_len)]
                    yes_no = True if word == var else False
                    if yes_no:
                        word_present += 1
                        print(i, "variation feature [{}] present in test data point [{}]".forr
                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(wo

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

In [65]:
```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_c

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).t
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_var
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variat
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fe

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_res
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseC
```

In [66]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respons
```

```
        print("(number of data points * number of features) in test data = ", test_x_responseC
        print("(number of data points * number of features) in cross validation data =", cv_x_
```

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

In [67]:
```
        print("One hot encoding features :")
        print("(number of data points * number of features) in train data = ", train_x_onehotC
        print("(number of data points * number of features) in test data = ", test_x_onehotCoC
        print("(number of data points * number of features) in cross validation data =", cv_x_
```

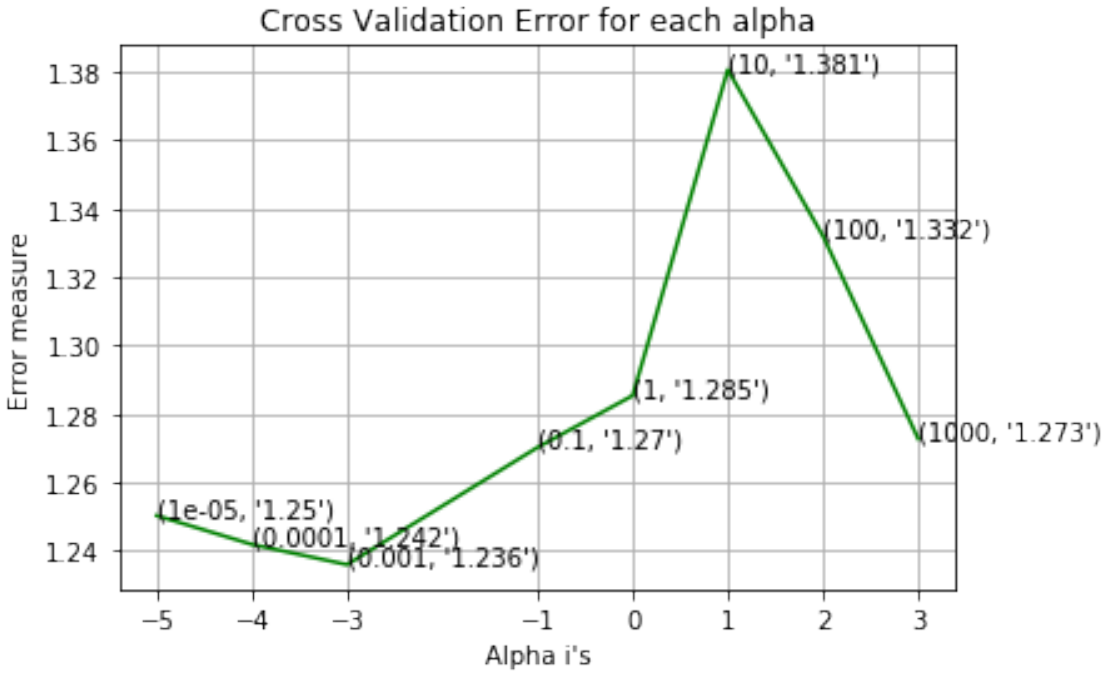
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 56215)
(number of data points * number of features) in test data =  (665, 56215)
(number of data points * number of features) in cross validation data = (532, 56215)
```

In [68]:
```
        # find more about Multinomial Naive base function here http://scikit-learn.org/stable/
        # -------------------------
        # default paramters
        # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

```
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=
    # to avoid rounding error while multiplying probabilites we use log-probability e
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 1e-05
Log Loss : 1.2502302619333674
for alpha = 0.0001
Log Loss : 1.2418143505886456
for alpha = 0.001
Log Loss : 1.235895867852822
for alpha = 0.1
Log Loss : 1.2701413590096187
for alpha = 1
```

```
Log Loss : 1.285300827097687
for alpha = 10
Log Loss : 1.3805905138736116
for alpha = 100
Log Loss : 1.3319540728168415
for alpha = 1000
Log Loss : 1.2727193422741074
```

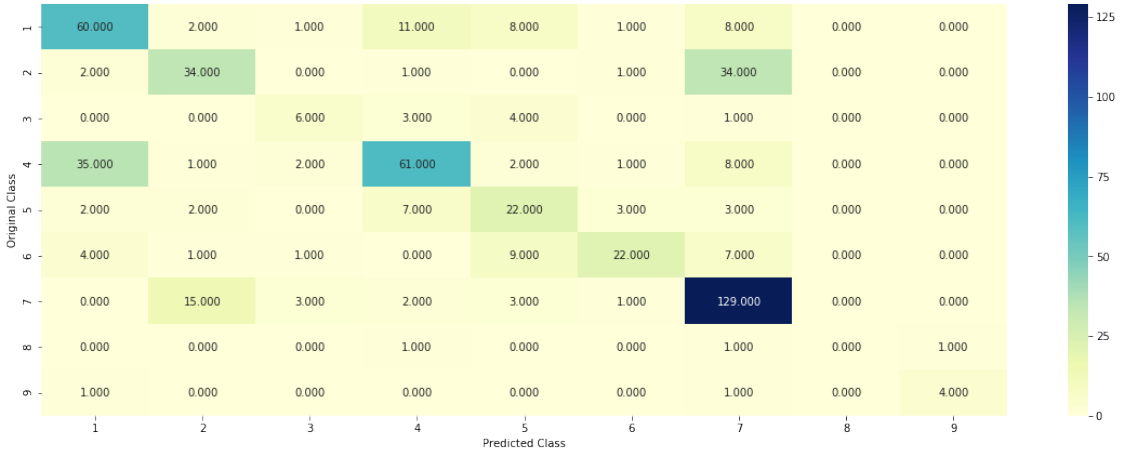



```
For values of best alpha =  0.001 The train log loss is: 0.8937387518752322
For values of best alpha =  0.001 The cross validation log loss is: 1.235895867852822
For values of best alpha =  0.001 The test log loss is: 1.2726713284432736
```

```
In [69]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/
         # -------------------------
         # default paramters
         # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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

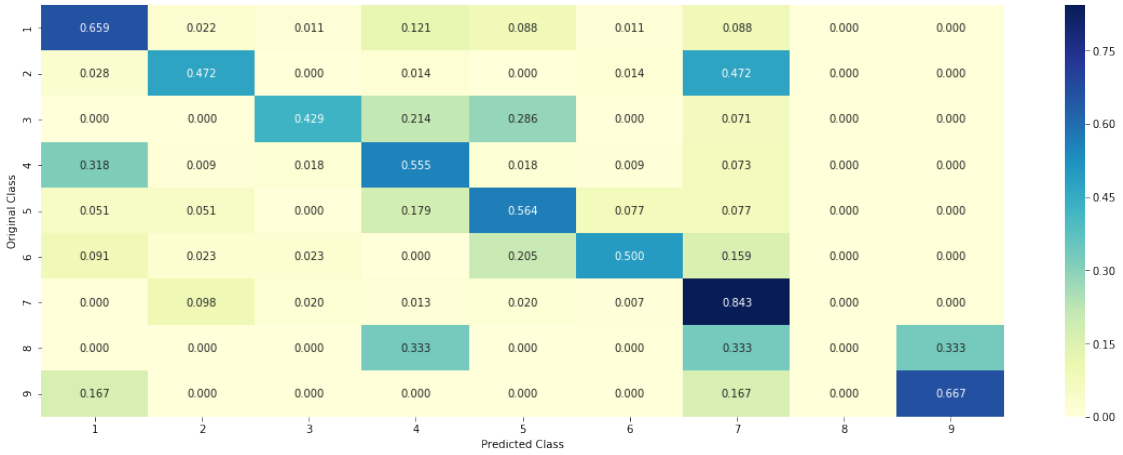
```
        # -----------------------


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

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

```
Log Loss : 1.2358958678528 22
Number of missclassified point : 0.36466165413533835
-------------------- Confusion matrix --------------------
```




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

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




```
In [70]: 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_onehotC
         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

Predicted Class : 7
Predicted Class Probabilities: [[0.086  0.0917 0.0131 0.1215 0.0362 0.0368 0.6065 0.0044 0.003
```

```
Actual Class : 7
--------------------------------------------------
17 Text feature [kinase] present in test data point [True]
18 Text feature [presence] present in test data point [True]
19 Text feature [inhibitor] present in test data point [True]
20 Text feature [independent] present in test data point [True]
22 Text feature [contrast] present in test data point [True]
23 Text feature [well] present in test data point [True]
24 Text feature [activating] present in test data point [True]
26 Text feature [potential] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [showed] present in test data point [True]
29 Text feature [compared] present in test data point [True]
30 Text feature [expressing] present in test data point [True]
31 Text feature [previously] present in test data point [True]
32 Text feature [cell] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [10] present in test data point [True]
35 Text feature [cells] present in test data point [True]
36 Text feature [obtained] present in test data point [True]
38 Text feature [shown] present in test data point [True]
40 Text feature [similar] present in test data point [True]
41 Text feature [found] present in test data point [True]
42 Text feature [suggest] present in test data point [True]
43 Text feature [activation] present in test data point [True]
44 Text feature [may] present in test data point [True]
45 Text feature [growth] present in test data point [True]
46 Text feature [treated] present in test data point [True]
47 Text feature [described] present in test data point [True]
48 Text feature [mutations] present in test data point [True]
49 Text feature [total] present in test data point [True]
50 Text feature [addition] present in test data point [True]
51 Text feature [inhibition] present in test data point [True]
54 Text feature [studies] present in test data point [True]
56 Text feature [inhibitors] present in test data point [True]
57 Text feature [various] present in test data point [True]
58 Text feature [respectively] present in test data point [True]
59 Text feature [12] present in test data point [True]
60 Text feature [report] present in test data point [True]
61 Text feature [reported] present in test data point [True]
62 Text feature [confirmed] present in test data point [True]
64 Text feature [including] present in test data point [True]
65 Text feature [increased] present in test data point [True]
66 Text feature [identified] present in test data point [True]
68 Text feature [concentrations] present in test data point [True]
69 Text feature [using] present in test data point [True]
72 Text feature [although] present in test data point [True]
73 Text feature [approximately] present in test data point [True]
```

```
74 Text feature [figure] present in test data point [True]
75 Text feature [proliferation] present in test data point [True]
77 Text feature [mutation] present in test data point [True]
78 Text feature [15] present in test data point [True]
80 Text feature [suggests] present in test data point [True]
81 Text feature [approved] present in test data point [True]
82 Text feature [two] present in test data point [True]
84 Text feature [sensitive] present in test data point [True]
85 Text feature [followed] present in test data point [True]
86 Text feature [due] present in test data point [True]
87 Text feature [whereas] present in test data point [True]
88 Text feature [different] present in test data point [True]
91 Text feature [activated] present in test data point [True]
92 Text feature [phosphorylation] present in test data point [True]
93 Text feature [molecular] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [small] present in test data point [True]
96 Text feature [25] present in test data point [True]
97 Text feature [measured] present in test data point [True]
98 Text feature [thus] present in test data point [True]
99 Text feature [another] present in test data point [True]
Out of the top  100  features  67 are present in query point
```

```
In [71]: 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_onehotC
         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
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.096  0.368  0.0147 0.1356 0.0403 0.0408 0.2954 0.005  0.004
Actual Class : 6
--------------------------------------------------
17 Text feature [identified] present in test data point [True]
20 Text feature [including] present in test data point [True]
21 Text feature [molecular] present in test data point [True]
23 Text feature [identify] present in test data point [True]
24 Text feature [using] present in test data point [True]
25 Text feature [patient] present in test data point [True]
26 Text feature [clinical] present in test data point [True]
27 Text feature [confirmed] present in test data point [True]
28 Text feature [another] present in test data point [True]
29 Text feature [identification] present in test data point [True]
```

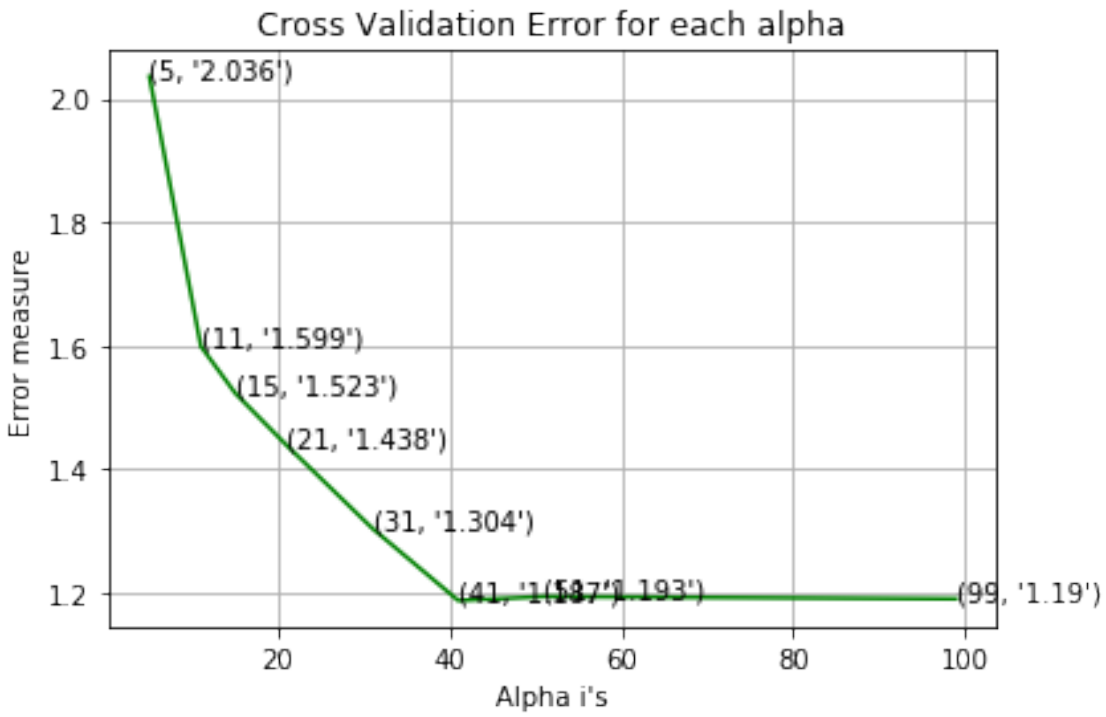
```
30 Text feature [new] present in test data point [True]
31 Text feature [potential] present in test data point [True]
32 Text feature [well] present in test data point [True]
33 Text feature [therapeutic] present in test data point [True]
34 Text feature [may] present in test data point [True]
35 Text feature [kinase] present in test data point [True]
38 Text feature [different] present in test data point [True]
39 Text feature [detection] present in test data point [True]
40 Text feature [mutations] present in test data point [True]
41 Text feature [revealed] present in test data point [True]
42 Text feature [therapy] present in test data point [True]
43 Text feature [harbor] present in test data point [True]
44 Text feature [present] present in test data point [True]
45 Text feature [found] present in test data point [True]
46 Text feature [go] present in test data point [True]
47 Text feature [tumor] present in test data point [True]
48 Text feature [previously] present in test data point [True]
51 Text feature [also] present in test data point [True]
52 Text feature [12] present in test data point [True]
53 Text feature [harboring] present in test data point [True]
54 Text feature [samples] present in test data point [True]
55 Text feature [specific] present in test data point [True]
56 Text feature [10] present in test data point [True]
57 Text feature [manufacturer] present in test data point [True]
58 Text feature [per] present in test data point [True]
59 Text feature [highly] present in test data point [True]
60 Text feature [studies] present in test data point [True]
61 Text feature [inhibitor] present in test data point [True]
62 Text feature [pcr] present in test data point [True]
63 Text feature [table] present in test data point [True]
65 Text feature [gene] present in test data point [True]
66 Text feature [similar] present in test data point [True]
68 Text feature [one] present in test data point [True]
69 Text feature [performed] present in test data point [True]
72 Text feature [recently] present in test data point [True]
73 Text feature [observed] present in test data point [True]
74 Text feature [mutated] present in test data point [True]
76 Text feature [cell] present in test data point [True]
79 Text feature [described] present in test data point [True]
80 Text feature [15] present in test data point [True]
81 Text feature [however] present in test data point [True]
85 Text feature [sample] present in test data point [True]
86 Text feature [findings] present in test data point [True]
87 Text feature [90] present in test data point [True]
88 Text feature [analysis] present in test data point [True]
90 Text feature [characterized] present in test data point [True]
91 Text feature [number] present in test data point [True]
92 Text feature [40] present in test data point [True]
```

```
94 Text feature [mutation] present in test data point [True]
95 Text feature [according] present in test data point [True]
96 Text feature [single] present in test data point [True]
Out of the top  100  features  61 are present in query point
```

In [72]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
# 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/


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

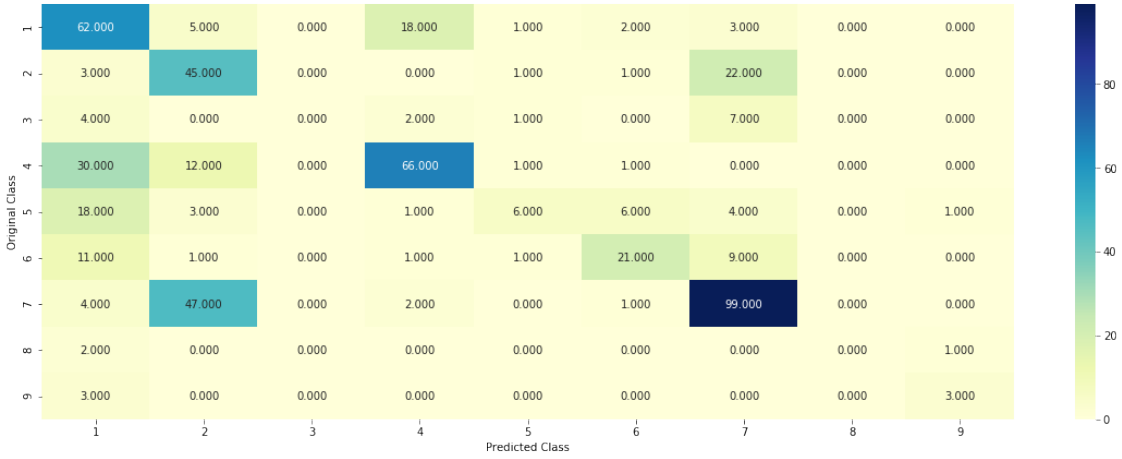
```
for alpha = 5
Log Loss : 2.035836009593536
for alpha = 11
Log Loss : 1.598711614546682
for alpha = 15
Log Loss : 1.52286115381139
for alpha = 21
Log Loss : 1.4383627328010935
for alpha = 31
Log Loss : 1.3037236766951033
for alpha = 41
Log Loss : 1.1869020777012569
for alpha = 51
Log Loss : 1.193260833090447
for alpha = 99
Log Loss : 1.1896762021597282
```



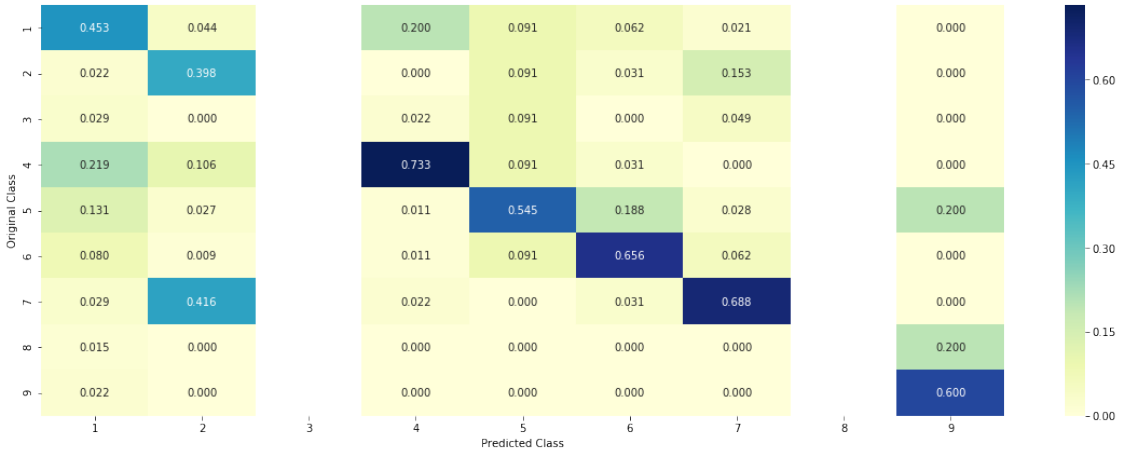

```
For values of best alpha =  41 The train log loss is: 0.7933849505177542
For values of best alpha =  41 The cross validation log loss is: 1.1869020777012569
For values of best alpha =  41 The test log loss is: 1.1602805998746166
```

```
In [73]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
         # ------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
         # 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/
         #-------------------------------------
         clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCodin
```

```
Log loss : 1.1869020777012569
Number of mis-classified points : 0.4323308270676692
-------------------- Confusion matrix --------------------
```



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



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


```
In [74]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 1
Actual Class : 7
The  41  nearest neighbours of the test points belongs to classes [2 2 3 7 7 3 3 7 7 7 7 7 7 7
 2 7 7 7]
Fequency of nearest points : Counter({7: 27, 3: 6, 2: 3, 5: 3, 1: 1, 4: 1})
```

```
In [75]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 100

         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
         print("Predicted Class :", predicted_cls[0])
         print("Actual Class :", test_y[test_point_index])
         neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alp
```

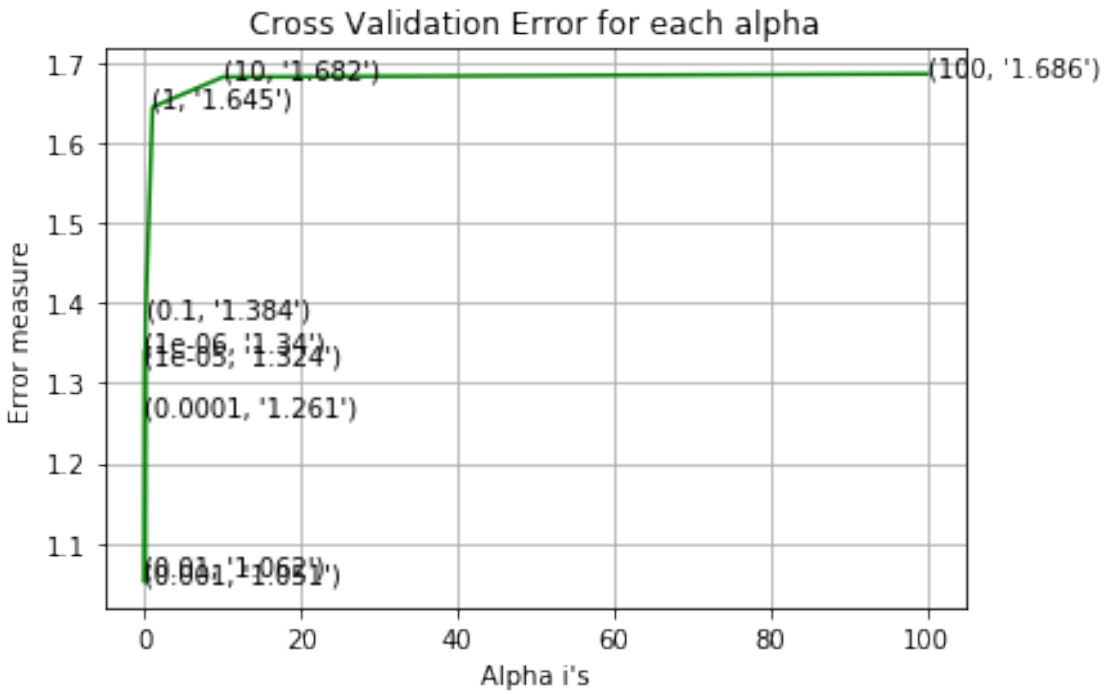
```
        print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the te
        print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 6
the k value for knn is 41 and the nearest neighbours of the test points belongs to classes [1 6
 1 4 4 4]
Fequency of nearest points : Counter({1: 19, 6: 10, 4: 8, 5: 4})
```

In [76]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ------------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# 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', ra
    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=
        # to avoid rounding error while multiplying probabilites we use log-probability e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


    best_alpha = np.argmin(cv_log_error_array)
    clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
    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_
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
    predict_y = sig_clf.predict_proba(test_x_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l
```

```
for alpha = 1e-06
Log Loss : 1.3396057863802107
for alpha = 1e-05
Log Loss : 1.323746794263415
for alpha = 0.0001
Log Loss : 1.260795085929115
for alpha = 0.001
Log Loss : 1.051020052605905
for alpha = 0.01
Log Loss : 1.0621739623078938
for alpha = 0.1
Log Loss : 1.3840894420205923
for alpha = 1
Log Loss : 1.6445490492388815
for alpha = 10
Log Loss : 1.6821585658681164
for alpha = 100
Log Loss : 1.6860072875746157
```

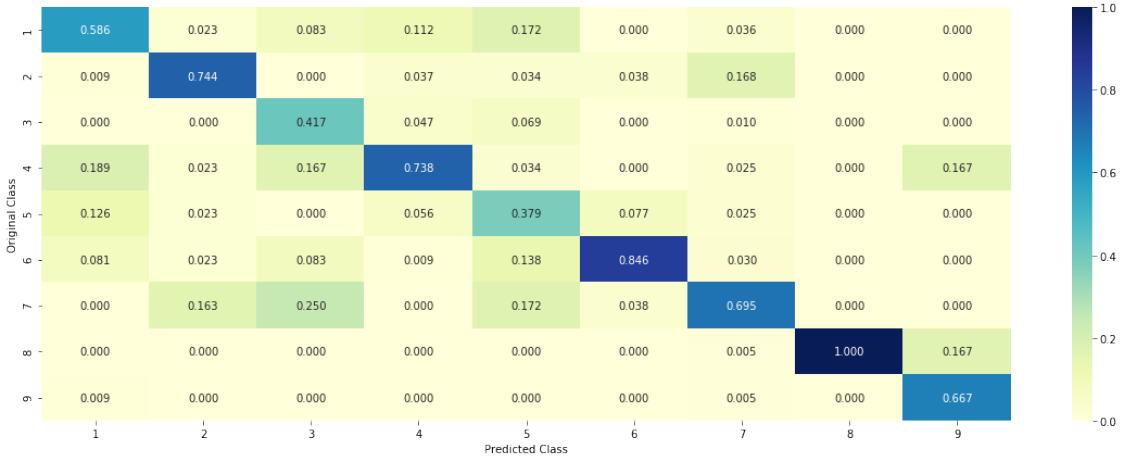



```
For values of best alpha =  0.001 The train log loss is: 0.6161886482677715
For values of best alpha =  0.001 The cross validation log loss is: 1.051020052605905
For values of best alpha =  0.001 The test log loss is: 1.0634882205959748
```

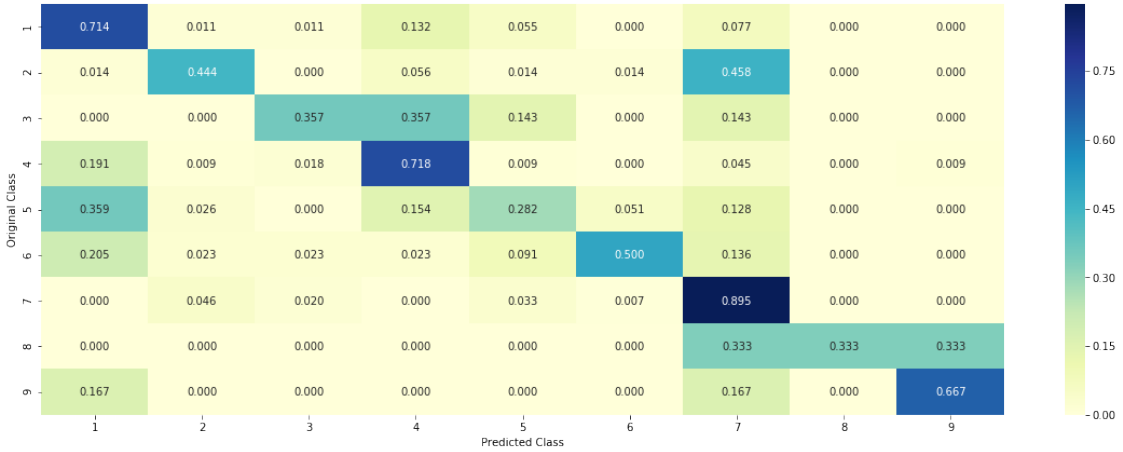
```
In [77]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # predict(X)        Predict class labels for samples in X.

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

```
Log loss : 1.051020052605905
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```

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



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



In [78]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

In [79]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
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_onehotCo
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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0343 0.0656 0.0127 0.0358 0.0169 0.0202 0.8045 0.0046 0.005
Actual Class : 7
--------------------------------------------------
17 Text feature [constitutively] present in test data point [True]
19 Text feature [constitutive] present in test data point [True]
94 Text feature [technology] present in test data point [True]
127 Text feature [ligand] present in test data point [True]
129 Text feature [tyrosyl] present in test data point [True]
139 Text feature [activated] present in test data point [True]
164 Text feature [oncogene] present in test data point [True]
252 Text feature [transformed] present in test data point [True]
253 Text feature [activation] present in test data point [True]
346 Text feature [independence] present in test data point [True]
369 Text feature [expressing] present in test data point [True]
372 Text feature [tyr] present in test data point [True]
397 Text feature [constants] present in test data point [True]
406 Text feature [benefits] present in test data point [True]
413 Text feature [inhibited] present in test data point [True]
485 Text feature [inhibitor] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

```
In [80]: 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_onehotC
         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
```

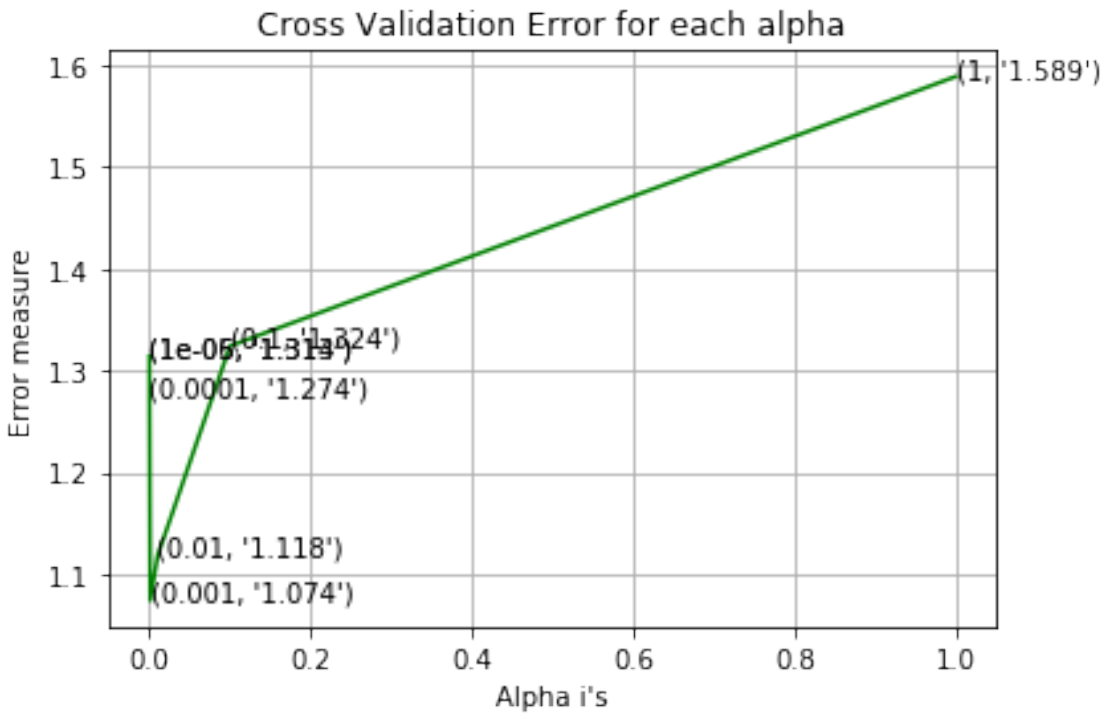
```
Predicted Class : 2
Predicted Class Probabilities: [[0.2161 0.2248 0.0169 0.0459 0.1882 0.1178 0.1694 0.009  0.011
Actual Class : 6
--------------------------------------------------
202 Text feature [rehovot] present in test data point [True]
259 Text feature [rptor] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

```
In [81]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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



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

```
for alpha = 1e-06
Log Loss : 1.3140453374154064
for alpha = 1e-05
Log Loss : 1.3134204127729872
for alpha = 0.0001
Log Loss : 1.2742241427243348
for alpha = 0.001
Log Loss : 1.073823832401177
for alpha = 0.01
Log Loss : 1.1175594155441337
for alpha = 0.1
Log Loss : 1.3241061547153685
for alpha = 1
Log Loss : 1.5890723664342818
```



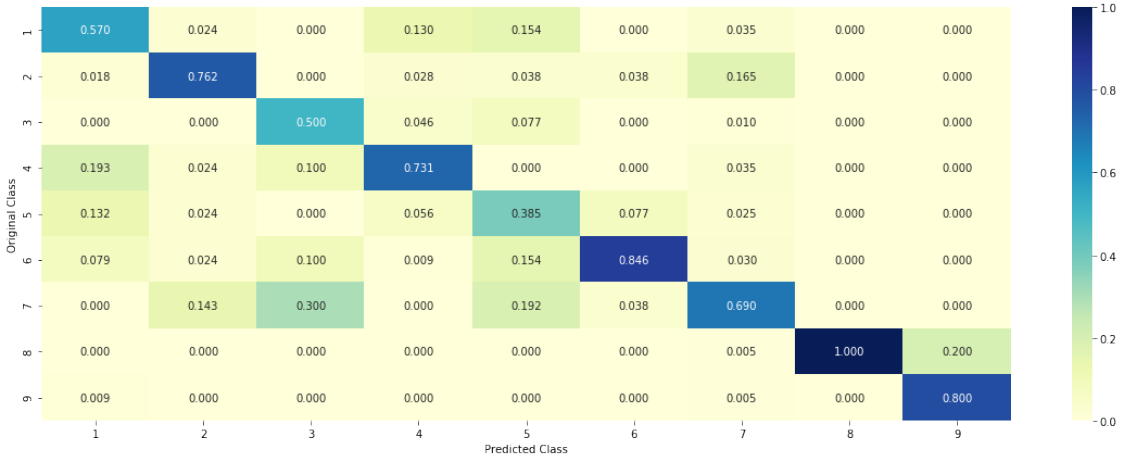

```
For values of best alpha =  0.001 The train log loss is: 0.6141356448792185
For values of best alpha =  0.001 The cross validation log loss is: 1.073823832401177
For values of best alpha =  0.001 The test log loss is: 1.0786890619983784
```

```
In [82]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # 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
```

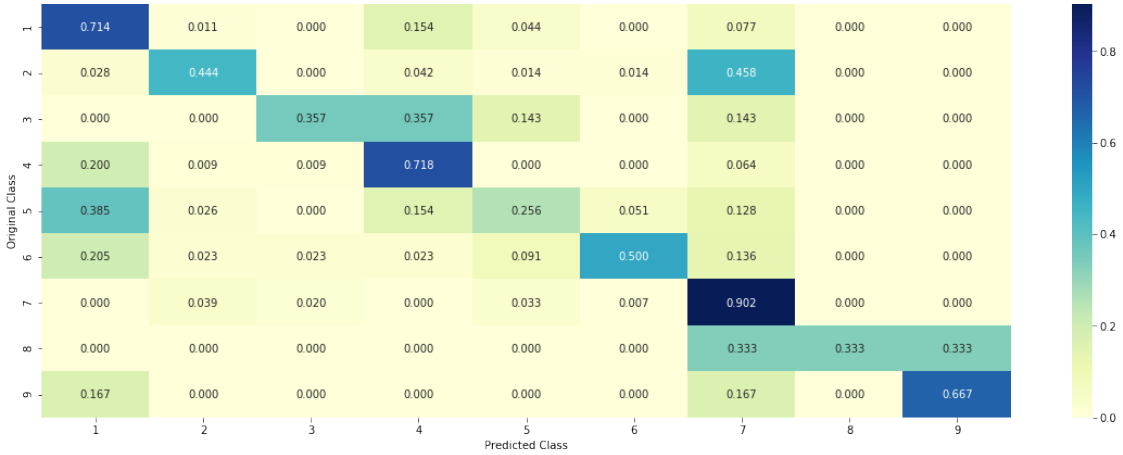
```
Log loss : 1.073823832401177
Number of mis-classified points : 0.3308270676691729
```

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

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



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




```
In [83]: 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_onehotCo
         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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0386 0.0723 0.0107 0.0401 0.0193 0.02   0.7907 0.0048 0.0034
Actual Class : 7
--------------------------------------------------
97 Text feature [constitutively] present in test data point [True]
108 Text feature [constitutive] present in test data point [True]
206 Text feature [technology] present in test data point [True]
279 Text feature [tyrosyl] present in test data point [True]
329 Text feature [isoforms] present in test data point [True]
344 Text feature [oncogene] present in test data point [True]
350 Text feature [activated] present in test data point [True]
355 Text feature [transformed] present in test data point [True]
388 Text feature [expressing] present in test data point [True]
442 Text feature [independence] present in test data point [True]
480 Text feature [ligand] present in test data point [True]
487 Text feature [benefits] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

```
In [84]: 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_onehotC
        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
```

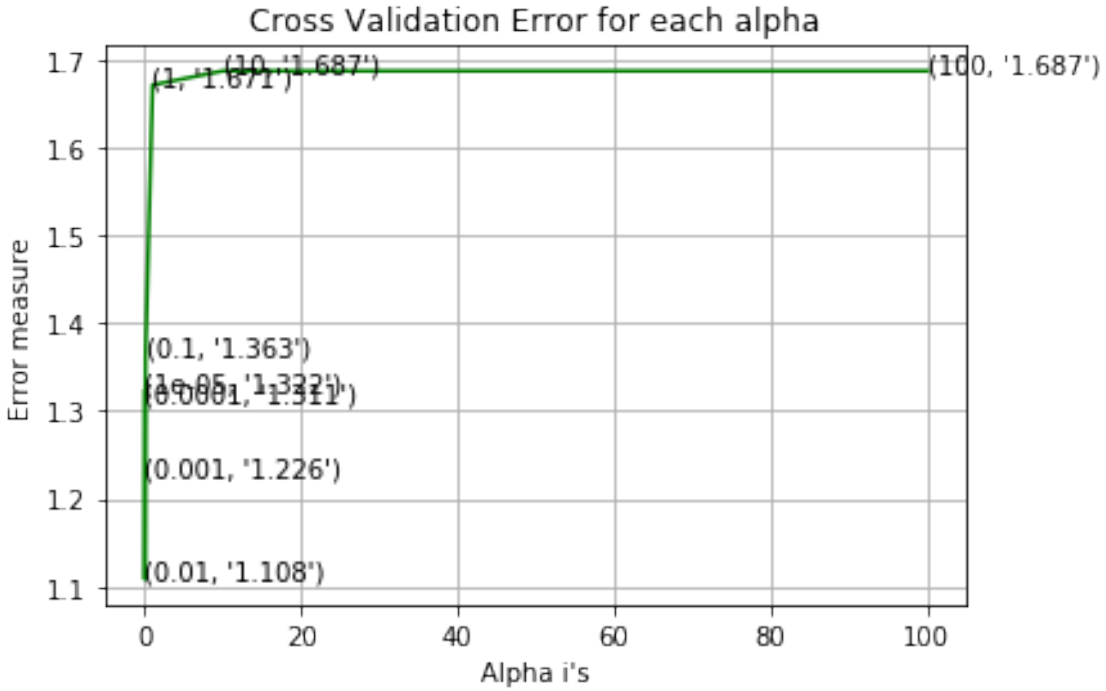
```
Predicted Class : 2
Predicted Class Probabilities: [[0.2194 0.2267 0.0105 0.0461 0.1789 0.1263 0.1782 0.0082 0.005
Actual Class : 6
--------------------------------------------------
214 Text feature [rehovot] present in test data point [True]
322 Text feature [rptor] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

In [85]:
```
        # read more about support vector machines with linear kernals here http://scikit-lear

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

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



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

        alpha = [10 ** x for x in range(-5, 3)]
```

```
        cv_log_error_array = []
        for i in alpha:
            print("for C =", i)
        #     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
            clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge'
            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=
            print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


        best_alpha = np.argmin(cv_log_error_array)
        # clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
        clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
        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_l
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(test_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for C = 1e-05
Log Loss : 1.3224048628952374
for C = 0.0001
Log Loss : 1.3105160643767317
for C = 0.001
Log Loss : 1.2255989333854782
for C = 0.01
Log Loss : 1.1082278080119787
for C = 0.1
Log Loss : 1.3630125393816452
for C = 1
```

```
Log Loss : 1.6707230281130523
for C = 10
Log Loss : 1.6866900997327665
for C = 100
Log Loss : 1.6866901069989726
```

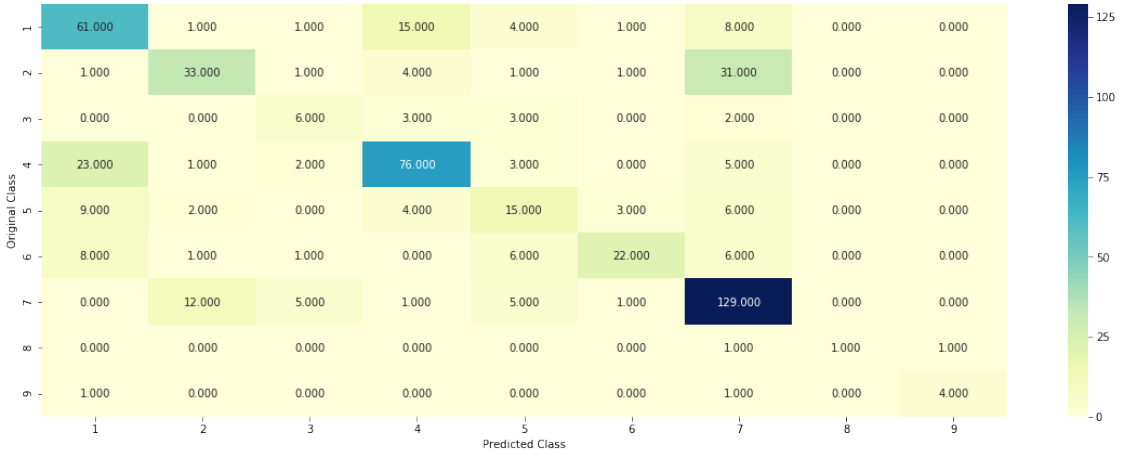



```
For values of best alpha =  0.01 The train log loss is: 0.7514155405830102
For values of best alpha =  0.01 The cross validation log loss is: 1.1082278080119787
For values of best alpha =  0.01 The test log loss is: 1.1211170002993518
```

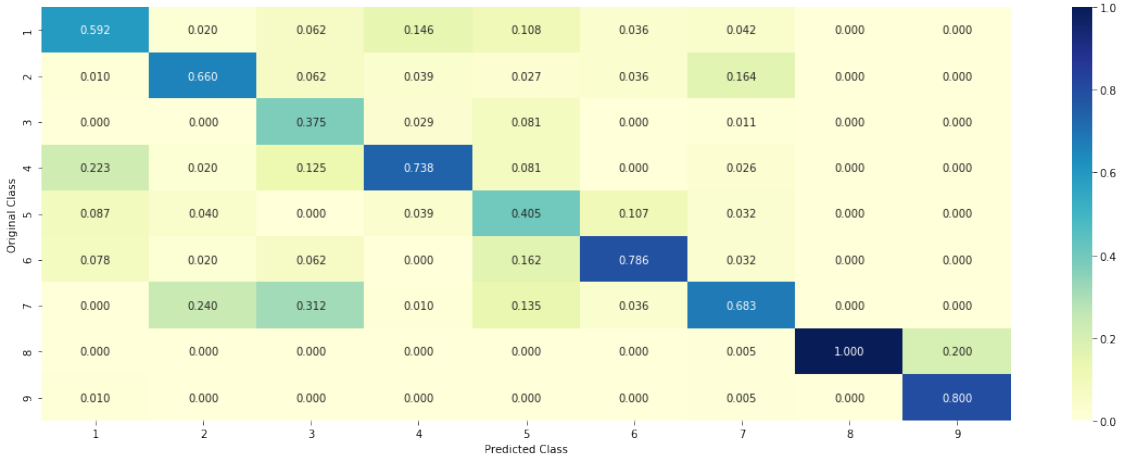
```
In [86]: # read more about support vector machines with linear kernals here http://scikit-lear

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

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

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

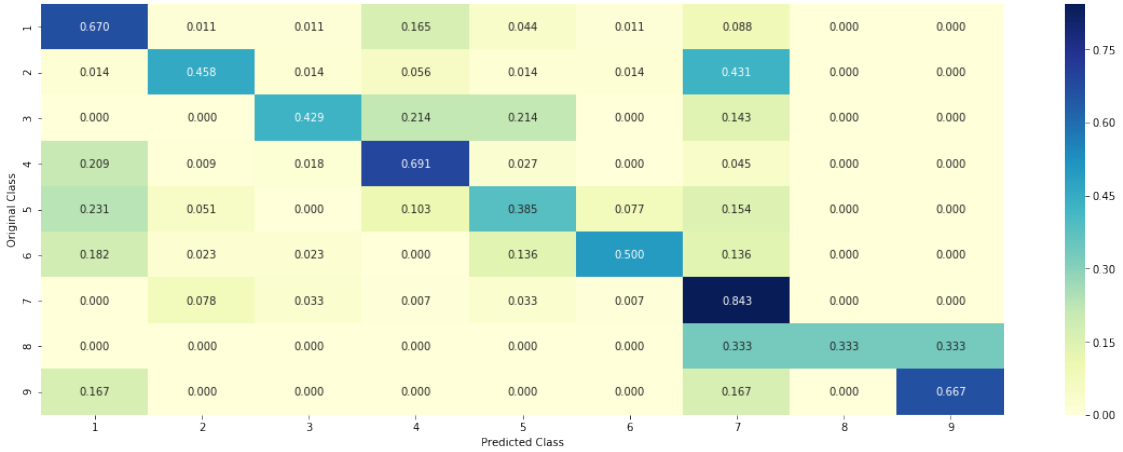
```
Log loss : 1.1082278080119787
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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




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




```
In [87]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
         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_onehotC
         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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0491 0.0445 0.0104 0.0625 0.035  0.0227 0.7666 0.0045 0.004
Actual Class : 7
--------------------------------------------------
29 Text feature [constitutively] present in test data point [True]
34 Text feature [tyrosyl] present in test data point [True]
39 Text feature [constitutive] present in test data point [True]
58 Text feature [technology] present in test data point [True]
77 Text feature [independence] present in test data point [True]
107 Text feature [sulfophenyl] present in test data point [True]
111 Text feature [expressing] present in test data point [True]
153 Text feature [benefits] present in test data point [True]
185 Text feature [activated] present in test data point [True]
187 Text feature [transformed] present in test data point [True]
193 Text feature [concentrations] present in test data point [True]
201 Text feature [ligand] present in test data point [True]
216 Text feature [activation] present in test data point [True]
226 Text feature [inhibited] present in test data point [True]
```

```
272 Text feature [oncogene] present in test data point [True]
277 Text feature [proliferation] present in test data point [True]
317 Text feature [phospho] present in test data point [True]
338 Text feature [nanomolar] present in test data point [True]
419 Text feature [interleukin] present in test data point [True]
430 Text feature [kinase] present in test data point [True]
447 Text feature [independently] present in test data point [True]
494 Text feature [activating] present in test data point [True]
Out of the top  500  features  22 are present in query point
```

```
In [88]: 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_onehotC
         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
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.14   0.2313 0.0179 0.0847 0.1486 0.1483 0.214  0.0069 0.008
Actual Class : 6
--------------------------------------------------
239 Text feature [bioview] present in test data point [True]
334 Text feature [ffpe] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

```
In [89]: # ---------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# 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, ra
        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_, e
        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_e
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',
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

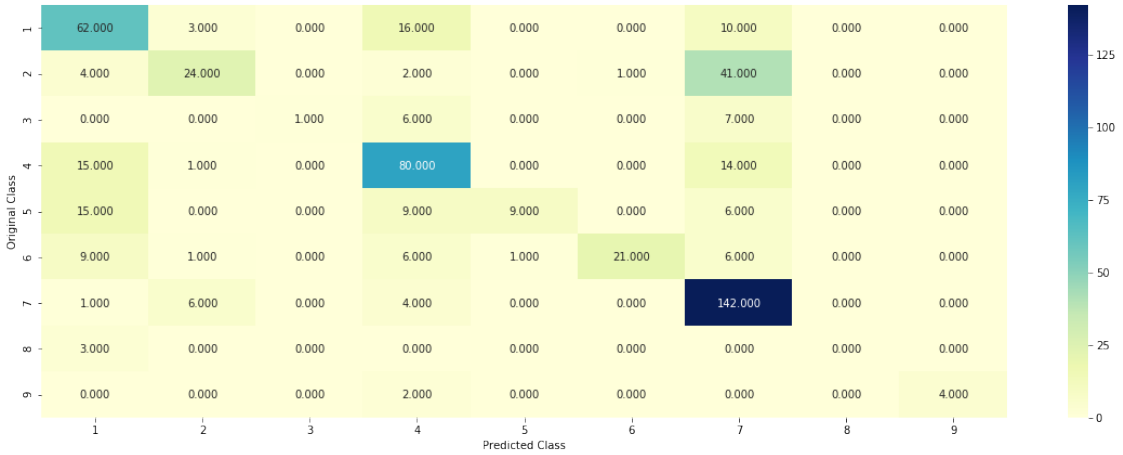
```
        predict_y = sig_clf.predict_proba(train_x_onehotCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validati
        predict_y = sig_clf.predict_proba(test_x_onehotCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2249797172001673
for n_estimators = 100 and max depth =  10
Log Loss : 1.1547784006813189
for n_estimators = 200 and max depth =  5
Log Loss : 1.2085205467333326
for n_estimators = 200 and max depth =  10
Log Loss : 1.1461692452328003
for n_estimators = 500 and max depth =  5
Log Loss : 1.204921767729624
for n_estimators = 500 and max depth =  10
Log Loss : 1.1401883319369728
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2052007207163573
for n_estimators = 1000 and max depth =  10
Log Loss : 1.137984095458224
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2060646141580522
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1363237003175484
For values of best estimator =  2000 The train log loss is: 0.7008965752247291
For values of best estimator =  2000 The cross validation log loss is: 1.1363237003175484
For values of best estimator =  2000 The test log loss is: 1.1670426881985698
```

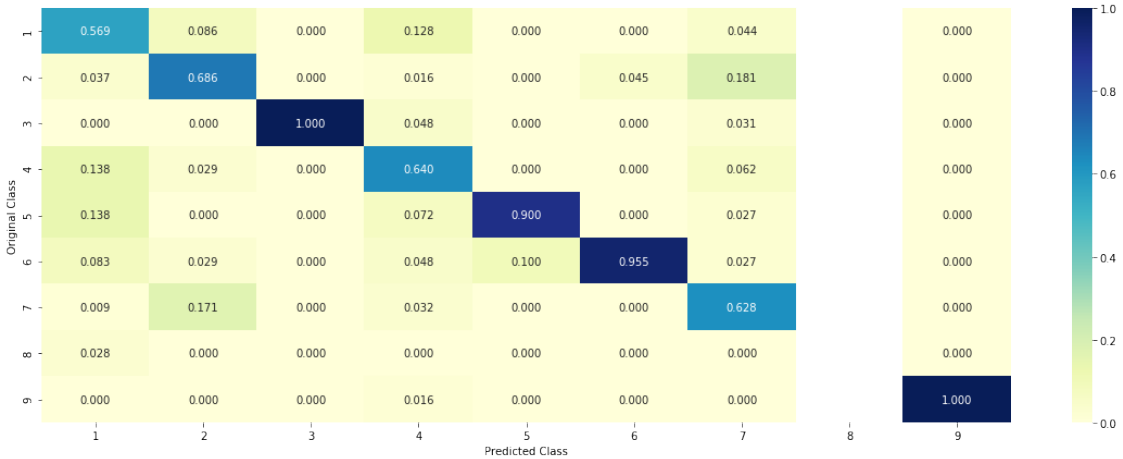
```
In [90]: # ---------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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

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

```
Log loss : 1.1363237003175484
Number of mis-classified points : 0.35526315789473684
--------------------- Confusion matrix ----------------------
```




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



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

In [91]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
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_onehotC
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0285 0.1194 0.0148 0.031  0.0342 0.027  0.7371 0.0042 0.003
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [constitutive] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [function] present in test data point [True]
```

```
10 Text feature [phosphorylation] present in test data point [True]
11 Text feature [inhibitor] present in test data point [True]
12 Text feature [signaling] present in test data point [True]
17 Text feature [growth] present in test data point [True]
18 Text feature [drug] present in test data point [True]
20 Text feature [inhibition] present in test data point [True]
21 Text feature [months] present in test data point [True]
23 Text feature [therapy] present in test data point [True]
24 Text feature [trials] present in test data point [True]
25 Text feature [resistance] present in test data point [True]
26 Text feature [oncogenic] present in test data point [True]
27 Text feature [constitutively] present in test data point [True]
28 Text feature [kinases] present in test data point [True]
30 Text feature [cells] present in test data point [True]
34 Text feature [receptor] present in test data point [True]
38 Text feature [patients] present in test data point [True]
39 Text feature [nsclc] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
48 Text feature [advanced] present in test data point [True]
49 Text feature [treated] present in test data point [True]
51 Text feature [ic50] present in test data point [True]
54 Text feature [proliferation] present in test data point [True]
56 Text feature [imatinib] present in test data point [True]
59 Text feature [protein] present in test data point [True]
64 Text feature [clinical] present in test data point [True]
66 Text feature [phospho] present in test data point [True]
71 Text feature [response] present in test data point [True]
72 Text feature [sensitive] present in test data point [True]
73 Text feature [cell] present in test data point [True]
75 Text feature [potency] present in test data point [True]
79 Text feature [autophosphorylation] present in test data point [True]
80 Text feature [independence] present in test data point [True]
90 Text feature [expression] present in test data point [True]
91 Text feature [tki] present in test data point [True]
96 Text feature [sensitivity] present in test data point [True]
97 Text feature [lines] present in test data point [True]
99 Text feature [phosphorylated] present in test data point [True]
Out of the top  100  features  46 are present in query point
```

```
In [92]: 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_onehotC
         print("Actuall Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
```

```
        get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],tes

Predicted Class : 1
Predicted Class Probabilities: [[0.245  0.2293 0.0243 0.1923 0.0678 0.0616 0.1629 0.0079 0.008
Actuall Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [function] present in test data point [True]
11 Text feature [inhibitor] present in test data point [True]
13 Text feature [suppressor] present in test data point [True]
15 Text feature [brca1] present in test data point [True]
17 Text feature [growth] present in test data point [True]
23 Text feature [therapy] present in test data point [True]
30 Text feature [cells] present in test data point [True]
34 Text feature [receptor] present in test data point [True]
38 Text feature [patients] present in test data point [True]
40 Text feature [stability] present in test data point [True]
45 Text feature [variants] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
59 Text feature [protein] present in test data point [True]
63 Text feature [functional] present in test data point [True]
64 Text feature [clinical] present in test data point [True]
65 Text feature [egfr] present in test data point [True]
68 Text feature [therapeutic] present in test data point [True]
71 Text feature [response] present in test data point [True]
72 Text feature [sensitive] present in test data point [True]
73 Text feature [cell] present in test data point [True]
74 Text feature [variant] present in test data point [True]
81 Text feature [brca2] present in test data point [True]
90 Text feature [expression] present in test data point [True]
96 Text feature [sensitivity] present in test data point [True]
97 Text feature [lines] present in test data point [True]
Out of the top  100  features  29 are present in query point
```

```
In [93]: # ---------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

```
# predict_proba (X)        Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# 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, ra
        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_, e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_e
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
```

```
        plt.show()
        '''

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

        predict_y = sig_clf.predict_proba(train_x_responseCoding)
        print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is
        predict_y = sig_clf.predict_proba(cv_x_responseCoding)
        print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation l
        predict_y = sig_clf.predict_proba(test_x_responseCoding)
        print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:"
```

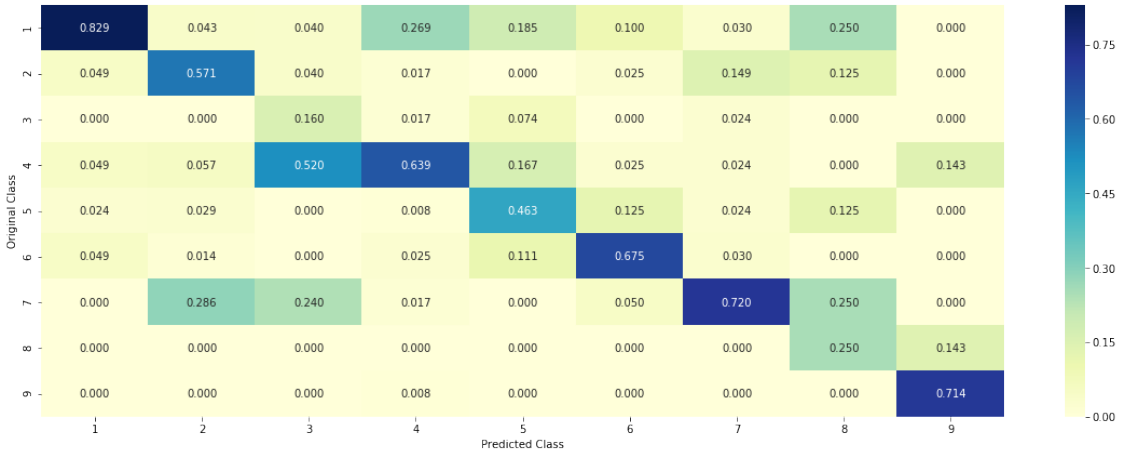
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2754538271140827
for n_estimators = 10 and max depth =  3
Log Loss : 1.8060159348865163
for n_estimators = 10 and max depth =  5
Log Loss : 1.5390997099575865
for n_estimators = 10 and max depth =  10
Log Loss : 2.107697366325483
for n_estimators = 50 and max depth =  2
Log Loss : 1.8270490497914469
for n_estimators = 50 and max depth =  3
Log Loss : 1.452203431486274
for n_estimators = 50 and max depth =  5
Log Loss : 1.3074840813865372
for n_estimators = 50 and max depth =  10
Log Loss : 1.8131186627665334
for n_estimators = 100 and max depth =  2
Log Loss : 1.638538595755599
for n_estimators = 100 and max depth =  3
Log Loss : 1.4370671424803463
for n_estimators = 100 and max depth =  5
Log Loss : 1.2034340750831225
for n_estimators = 100 and max depth =  10
Log Loss : 1.7546352139098231
for n_estimators = 200 and max depth =  2
Log Loss : 1.701605188728429
for n_estimators = 200 and max depth =  3
Log Loss : 1.4370328221462692
for n_estimators = 200 and max depth =  5
Log Loss : 1.3063851965433704
for n_estimators = 200 and max depth =  10
Log Loss : 1.7675657708074501
```

```
for n_estimators = 500 and max depth =  2
Log Loss : 1.6825584817138555
for n_estimators = 500 and max depth =  3
Log Loss : 1.499758588374824
for n_estimators = 500 and max depth =  5
Log Loss : 1.353177670148196
for n_estimators = 500 and max depth =  10
Log Loss : 1.8355506110630786
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6520569849400182
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4946927538504644
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3397726028068202
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8308103649767418
For values of best alpha =  100 The train log loss is: 0.048706567975859004
For values of best alpha =  100 The cross validation log loss is: 1.2034340750831225
For values of best alpha =  100 The test log loss is: 1.2571702623708587
```

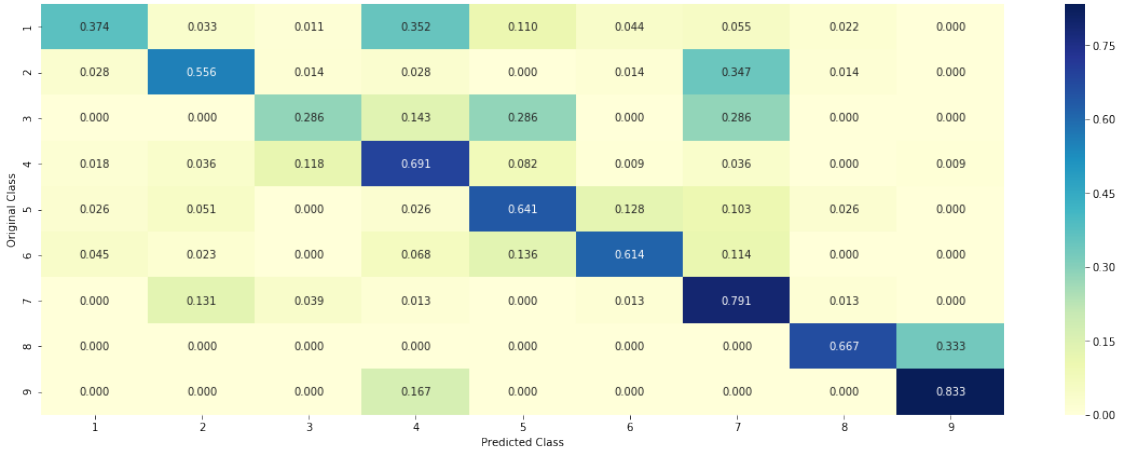
```
In [94]: # ---------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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

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

```
Log loss : 1.2034340750831225
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

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



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



In [95]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
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_response
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: [[0.0086 0.1875 0.262  0.0142 0.0288 0.0378 0.4219 0.0332 0.0058
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
```

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

```
In [96]: 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_response
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         for i in indices:
             if i<9:
                 print("Gene is important feature")
             elif i<18:
                 print("Variation is important feature")
             else:
                 print("Text is important feature")
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1193 0.0324 0.1284 0.1719 0.2009 0.1639 0.0163 0.1159 0.051
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
```

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

In [97]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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


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

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

```
# predict(X)        Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/

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

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

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

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_prob
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_p
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_or
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_classi
            sclf.fit(train_x_onehotCoding, train_y)
            print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_
            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: 1.05
Support vector machines : Log Loss: 1.67
Naive Bayes : Log Loss: 1.24
------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.042
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.525
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.091
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.153
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.432
```

```
In [98]: lr = LogisticRegression(C=0.1)
         sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=
         sclf.fit(train_x_onehotCoding, train_y)

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

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

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

         print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_onehot
         plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.6760186666342197
Log loss (CV) on the stacking classifier : 1.0912766747855878
Log loss (test) on the stacking classifier : 1.1104730243686831
Number of missclassified point : 0.3669172932330827
-------------------- Confusion matrix --------------------
```

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

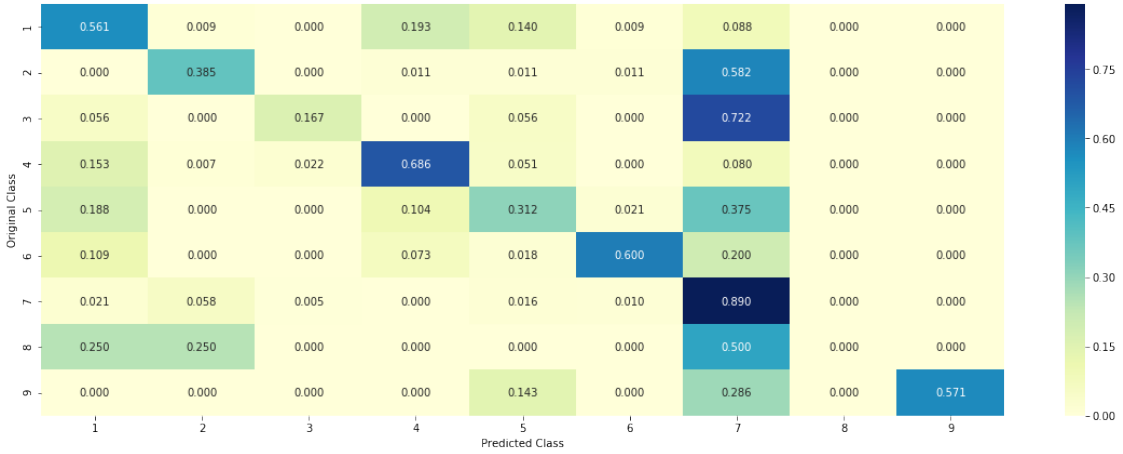
---------- Recall matrix (Row sum=1) ----------

In [99]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassi
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_cl
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_pro
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehot
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.9327927038569654
Log loss (CV) on the VotingClassifier : 1.178794786443973
Log loss (test) on the VotingClassifier : 1.2084503731970917
Number of missclassified point : 0.37142857142857144
-------------------- Confusion matrix --------------------
```

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


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




```
In [104]: train_gene_feature_onehotCoding

Out[104]: <2124x229 sparse matrix of type '<class 'numpy.int64'>'
                  with 2124 stored elements in Compressed Sparse Row format>

In [108]: train_df.to_pickle('train_df')  # where to save it, usually as a .pkl
          test_df.to_pickle('test_df')
          cv_df.to_pickle('cv_df')
          from scipy import sparse
```

```
sparse.save_npz("train_gene_feature_onehotCoding.npz", train_gene_feature_onehotCodi
sparse.save_npz("train_variation_feature_onehotCoding.npz", train_variation_feature_
sparse.save_npz("test_gene_feature_onehotCoding.npz", test_gene_feature_onehotCoding
sparse.save_npz("test_variation_feature_onehotCoding.npz", test_variation_feature_on
sparse.save_npz("cv_gene_feature_onehotCoding.npz", cv_gene_feature_onehotCoding)
sparse.save_npz("cv_variation_feature_onehotCoding.npz", cv_variation_feature_onehot
np.save('y_train',y_train)
np.save('y_test',y_test)
np.save('y_cv',y_cv)
```