Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

training_variants

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

training_text

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [3]:
```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

```
Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Out[3]:

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

training/training_variants is a comma separated file containing the description of the genetic mutations used for training.
Fields are

- **ID :** the id of the row used to link the mutation to the clinical evidence
- **Gene :** the gene where this genetic mutation is located
- **Variation :** the aminoacid change for this mutations
- **Class :** 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

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

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

Out[4]:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

3.1.3. Preprocessing of text

In [5]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [6]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_tim
e, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 236.790476 seconds
```

In [7]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[7]:

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

In [8]:
```
result[result.isnull().any(axis=1)]
```

Out[8]:

	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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	ID	Gene	Variation	Class	TEXT
2755	2755	BRAF	G596C	7	NaN

In [9]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result
['Variation']
```

In [10]:
```
result[result['ID']==1109]
```

Out[10]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [11]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

In [12]:
```
print('Number of data points in train data:', train_df.shape[0])
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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
```

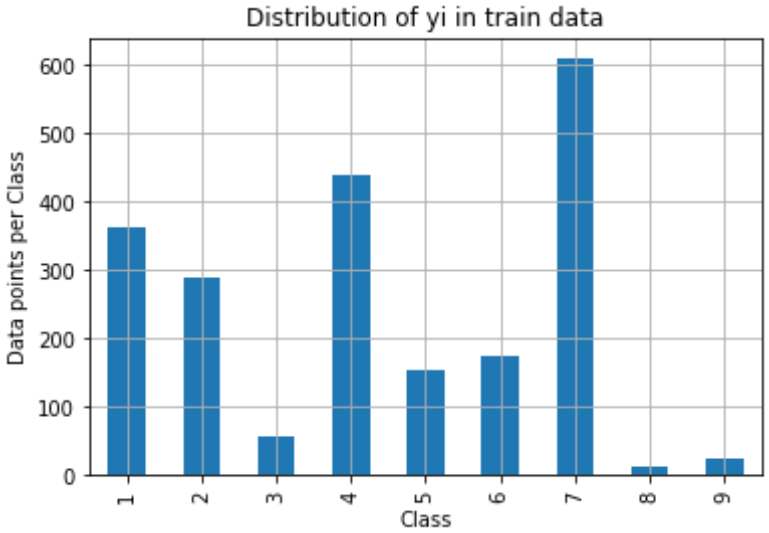
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [13]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
#from pandas.Series import sortlevel
train_class_distribution = train_df['Class'].value_counts().sort_index
()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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


print('-'*80)
```

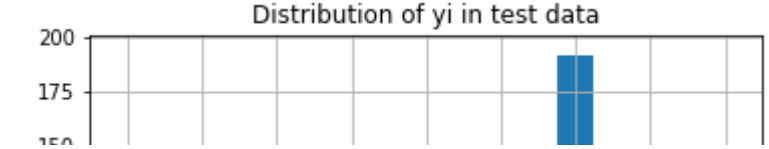
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

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

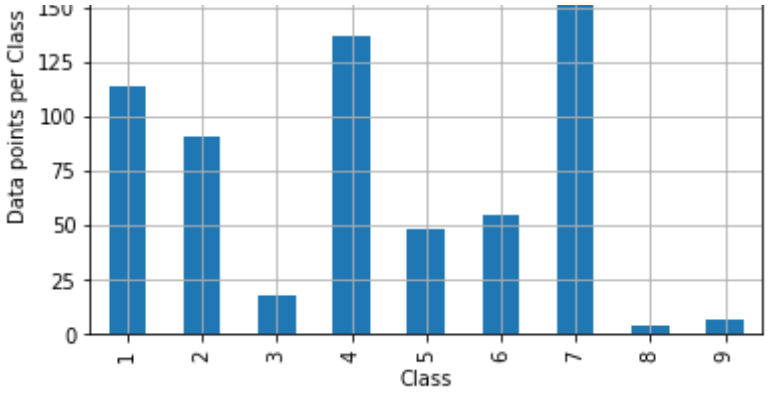
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



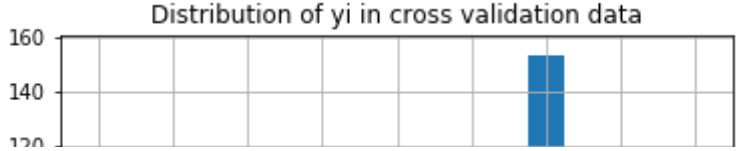

```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
------------------------------------------------------------------------
-----------
```



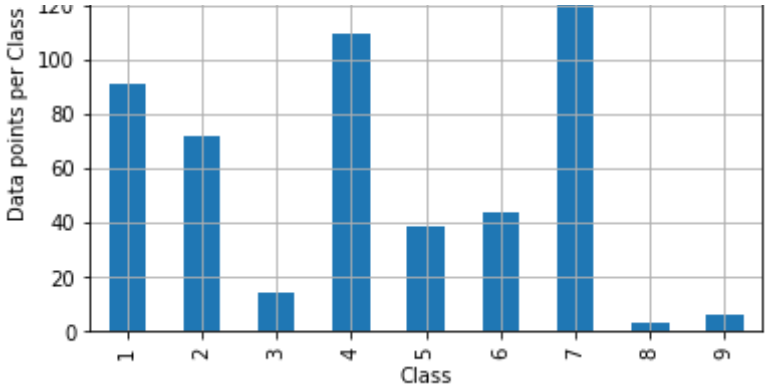
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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

3.2 Prediction using a 'Random' Model

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

In [14]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

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

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=la
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

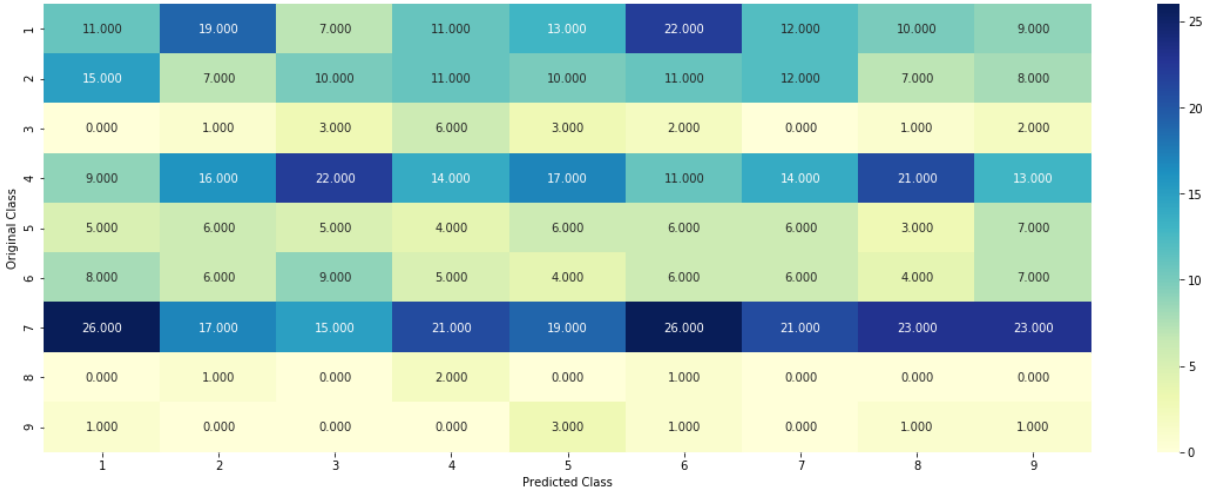
In [15]:

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

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


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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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 2.488829306065513
Log loss on Test Data using Random Model 2.4759421285304555
-------------------- Confusion matrix --------------------
```




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

3.3 Univariate Analysis

In [16]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_f
ea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.06818181818181818177, 0.06818181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.06060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for e
ach feature value in the data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [17]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 235
BRCA1     178
TP53      106
EGFR       90
PTEN       82
BRCA2      70
```

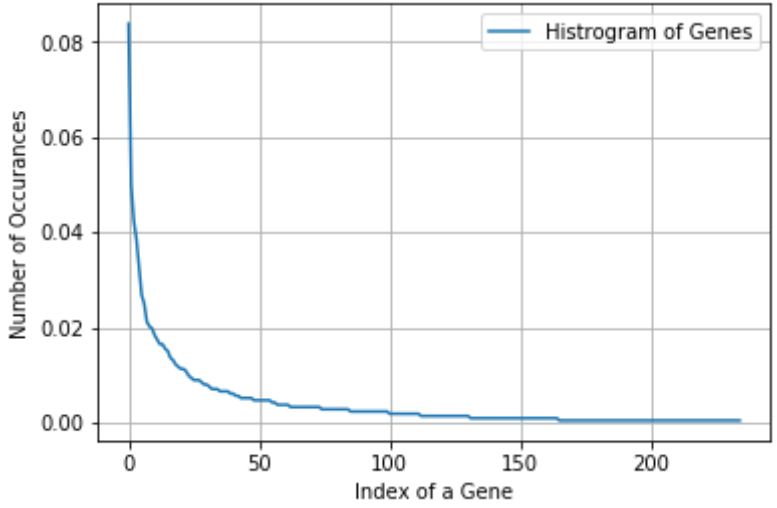
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
BRAF        57
KIT         53
ALK         45
PIK3CA      43
ERBB2       42
Name: Gene, dtype: int64
```

In [18]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

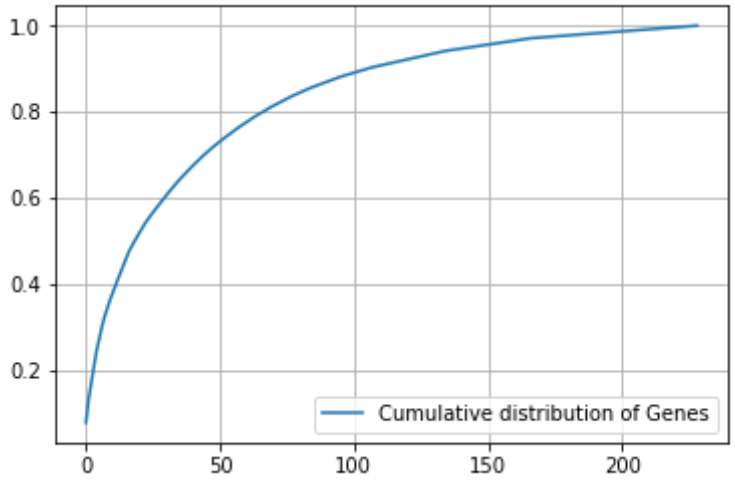
```
Ans: There are 235 different categories of genes in the train data, and
they are distibuted as follows
```

In [19]:
```
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()
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [22]:
```
# one-hot encoding of Gene feature.
from sklearn.feature_extraction.text import TfidfVectorizer
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
3113    RAD51C
1672      FLT3
2523     BRCA1
298      CHEK2
2729      BRAF
Name: Gene, dtype: object
```

In [24]:
```
gene_vectorizer.get_feature_names()
```

Out[24]:
```
['abl1',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                'acvr1',
                'ago2',
                'akt1',
                'akt2',
                'akt3',
                'alk',
                'apc',
                'ar',
                'araf',
                'arid1b',
                'arid2',
                'arid5b',
                'asxl1',
                'atm',
                'atr',
                'atrx',
                'aurka',
                'aurkb',
                'axin1',
                'axl',
                'b2m',
                'bap1',
                'bcl10',
                'bcl2',
                'bcl2l11',
                'bcor',
                'braf',
                'brca1',
                'brca2',
                'brd4',
                'brip1',
                'btk',
                'card11',
                'carm1',
                'casp8',
                'cbl',
                'ccnd1',
                'ccnd3',
                'ccne1',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                'cdh1',
                'cdk12',
                'cdk4',
                'cdk6',
                'cdk8',
                'cdkn1a',
                'cdkn1b',
                'cdkn2a',
                'cdkn2b',
                'cebpa',
                'chek2',
                'cic',
                'crebbp',
                'ctcf',
                'ctla4',
                'ctnnb1',
                'ddr2',
                'dicer1',
                'dnmt3a',
                'dnmt3b',
                'dusp4',
                'egfr',
                'elf3',
                'ep300',
                'epas1',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc4',
                'erg',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fam58a',
                'fanca',
                'fancc',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                'fat1',
                'fbxw7',
                'fgf3',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxp1',
                'fubp1',
                'gata3',
                'gli1',
                'gnaq',
                'gnas',
                'h3f3a',
                'hist1h1c',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'ikbke',
                'ikzf1',
                'il7r',
                'jak1',
                'jak2',
                'jun',
                'kdm5c',
                'kdr',
                'keap1',
                'kit',
                'kmt2a',
                'kmt2b',
                'kmt2c',
                'kmt2d',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                'knstrn',
                'kras',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'mdm2',
                'mdm4',
                'med12',
                'mef2b',
                'men1',
                'met',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'ncor1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'notch2',
                'npm1',
                'nras',
                'nsd1',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pbrm1',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pim1',
                'pms2',
                'pole',
                'ppm1d',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
                'rab35',
                'rac1',
                'rad21',
                'rad50',
                'rad51b',
                'rad51c',
                'rad51d',
                'rad54l',
                'raf1',
                'rasa1',
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rictor',
                'rit1',
                'ros1',
                'runx1',
                'rxra',
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
'rybp',
'sdhb',
'setd2',
'sf3b1',
'shoc2',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stat3',
'stk11',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'whsc1',
'whsc1l1',
'xpo1',
'xrcc2',
'yap1']
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [25]:
```
print("The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
The shape of gene feature: (2124, 234)
```

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

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

In [26]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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


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

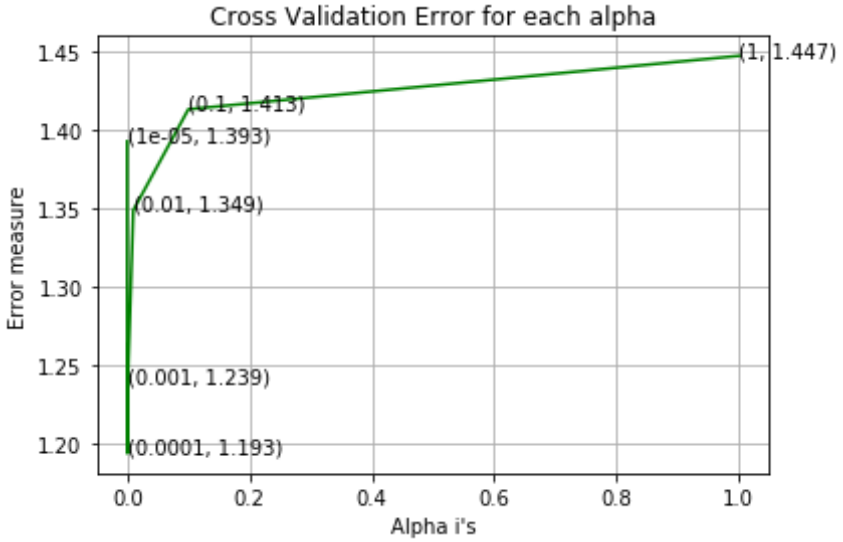
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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


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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
For values of alpha =  1e-05 The log loss is: 1.3927809380069236
For values of alpha =  0.0001 The log loss is: 1.1934563533960862
For values of alpha =  0.001 The log loss is: 1.2393314605080306
For values of alpha =  0.01 The log loss is: 1.3487862631612588


For values of alpha =  0.1 The log loss is: 1.4133043674520775
For values of alpha =  1 The log loss is: 1.4474645938183466
```




```
For values of best alpha =  0.0001 The train log loss is: 1.04413889395
57645
For values of best alpha =  0.0001 The cross validation log loss is: 1.
1934563533960862
For values of best alpha =  0.0001 The test log loss is: 1.224234647065
304
```

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

In [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by
 the ", unique_genes.shape[0], " genes in train dataset?")

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  23
5  genes in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 516 out of  532 : 96.99248120300751
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [28]:
```
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 : 1916
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

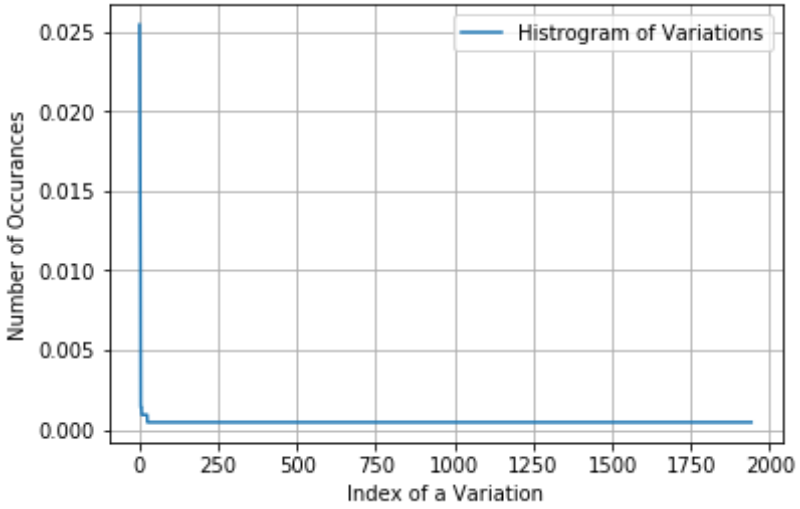
```
Truncating_Mutations    62
Amplification           57
Deletion                48
Fusions                 18
Overexpression           4
E17K                     3
T58I                     3
T167A                    2
C618R                    2
ETV6-NTRK3_Fusion        2
Name: Variation, dtype: int64
```

In [29]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

```
Ans: There are 1916 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [ ]:
```
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()
```

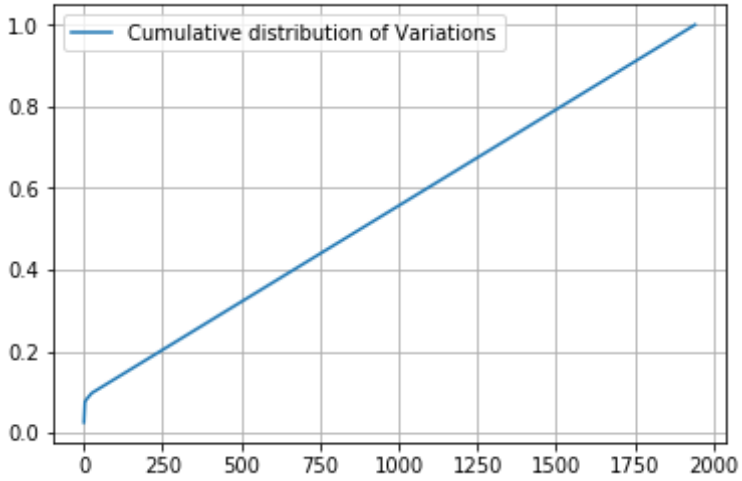
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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

```
[0.02542373 0.04613936 0.06544256 ... 0.99905838 0.99952919 1.        ]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [30]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [31]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [32]:
```
# tfidf  of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_tfidf = variation_vectorizer.fit_transform(trai
n_df['Variation'])
test_variation_feature_tfidf = variation_vectorizer.transform(test_df[
'Variation'])
cv_variation_feature_tfidf = variation_vectorizer.transform(cv_df['Vari
ation'])
```

In [33]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_tfidf.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature: (2124, 1943)
```

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

Let's build a model just like the earlier!

In [34]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

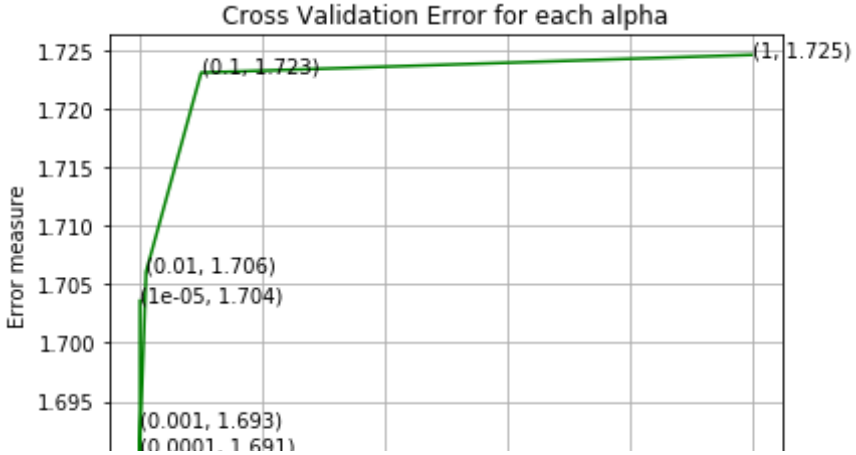
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)

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

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

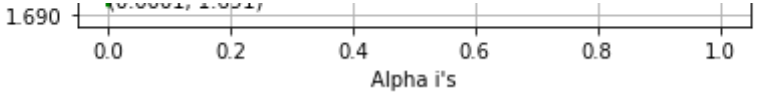
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_tfidf, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.703597964713451
For values of alpha =  0.0001 The log loss is: 1.6906535785540606
For values of alpha =  0.001 The log loss is: 1.6929148079457572
For values of alpha =  0.01 The log loss is: 1.7060728844696305
For values of alpha =  0.1 The log loss is: 1.7230721960672617
For values of alpha =  1 The log loss is: 1.724592370370146
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
For values of best alpha =  0.0001 The train log loss is: 0.76516790567
71195
For values of best alpha =  0.0001 The cross validation log loss is: 1.

6906535785540606
For values of best alpha =  0.0001 The test log loss is: 1.720250190430
4088
```

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

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

In [35]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1916  genes in test and
cross validation data sets?
Ans
1. In test data 59 out of 665 : 8.87218045112782
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?

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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 [36]:
```
# 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 [37]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [38]:
```
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
train_text_feature_tfidf = text_vectorizer.fit_transform(train_df['TEX
T'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [39]:

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

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


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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

In [40]:
```
#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 [41]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [42]:
```
# don't forget to normalize every feature
train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)

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

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

In [43]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [44]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Counter({255.28939113127436: 1, 181.80617044327732: 1, 139.789282429549
2: 1, 132.4849375478882: 1, 129.65472810172898: 1, 119.44676626531621:
1, 118.83385302402942: 1, 115.90754720365484: 1, 108.89111548503517: 1,
107.97600790342364: 1, 107.51891966464089: 1, 89.77257925296864: 1, 89.
0788668577643: 1, 87.90056923508979: 1, 80.86628676375747: 1, 80.819291
2787149: 1, 80.01222063468957: 1, 79.3843636625463: 1, 77.9913096086347
5: 1, 77.2232976429105: 1, 76.23465828542963: 1, 74.99164879290615: 1,
70.39969495124286: 1, 69.54753094430285: 1, 67.4796310039614: 1, 67.361
95870779648: 1, 67.35836954133627: 1, 65.41344864774236: 1, 64.98539534
505252: 1, 63.772211983808155: 1, 63.311555568505696: 1, 63.23564158104
8654: 1, 62.65409180314159: 1, 58.612497000521245: 1, 57.83728735167872
5: 1, 56.9842459115468: 1, 56.36070089589672: 1, 56.204314585281985: 1,
53.906093165077074: 1, 50.72446299997254: 1, 50.13798101791186: 1, 49.1
0770600688646: 1, 48.85947547478954: 1, 48.77032537982006: 1, 48.229166
3997258: 1, 46.35948448489216: 1, 45.454591740165455: 1, 44.53664091701
4466: 1, 44.079579599437324: 1, 43.80619109618225: 1, 43.5369532481201
4: 1, 43.51284152518571: 1, 43.39605729444797: 1, 42.86525727434377: 1,
42.555003137895156: 1, 42.02510844811882: 1, 41.82600579626675: 1, 41.8
1411042871249: 1, 41.573313518647836: 1, 41.549255017984905: 1, 41.4929
62060665164: 1, 41.282817001209025: 1, 40.96313413136087: 1, 40.9342841
13164736: 1, 40.31368395709789: 1, 40.030857491489826: 1, 39.5724730741
3352: 1, 39.485467916147726: 1, 39.39302070917543: 1, 39.3016449988036
4: 1, 38.90965105157892: 1, 38.86159910921608: 1, 38.214841096151886:
1, 37.40469859978633: 1, 37.11190030514064: 1, 36.32083504862765: 1, 3
6.24755243849658: 1, 36.11001498585434: 1, 35.781301888466935: 1, 35.59
587313455213: 1, 35.42560186439317: 1, 35.355284503377284: 1, 35.155154
0768233: 1, 35.09634297101238: 1, 34.80303798253792: 1, 34.610121774664
705: 1, 34.54254792636133: 1, 34.352763815767595: 1, 33.94108861198326:
1, 33.74358930942616: 1, 33.728208644750865: 1, 33.21212263744431: 1, 3
3.192471000371604: 1, 33.185934081242316: 1, 32.960736158463575: 1, 32.
960365679581294: 1, 32.77651806111244: 1, 32.42872310802117: 1, 32.2168
7494169435: 1, 32.20121307715539: 1, 32.14523733785601: 1, 31.916760099
931693: 1, 31.805959187479946: 1, 31.789896781279523: 1, 31.68314626351
9923: 1, 31.592078296754416: 1, 31.58686918722041: 1, 31.5842021705308
3: 1, 31.54015775394874: 1, 31.436274662868733: 1, 31.10471538844034:
1, 30.98902538034215: 1, 30.910687829505086: 1, 30.900579008558093: 1,
30.847525025660747: 1, 30.770892070561644: 1, 30.666279652601443: 1, 3
0.638218676452755: 1, 30.606274518991572: 1, 30.2152800746225: 1, 30.02
3525720517686: 1, 29.762263536425767: 1, 29.63060238947793: 1, 29.54206
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4633344175: 1, 29.1882336368097: 1, 29.095133616100465: 1, 28.811158597
46381: 1, 28.7189937950662: 1, 28.57143322736293: 1, 28.49702733444103
2: 1, 28.44929987254349: 1, 28.010005584161984: 1, 27.85849316141703:
1, 27.81308860853495: 1, 27.66927483470304: 1, 27.42148606992333: 1, 2
7.390009995550667: 1, 27.362362150213723: 1, 27.34790365546813: 1, 27.2
05211950999693: 1, 27.171818902493964: 1, 27.151726130168466: 1, 26.876
204080398693: 1, 26.787611150296907: 1, 26.552913172331753: 1, 26.53438
7722186672: 1, 26.5088233992274: 1, 26.34058670249068: 1, 26.1183244718
18944: 1, 25.958528222156072: 1, 25.864384815538134: 1, 25.794464927001
35: 1, 25.75270070795258: 1, 25.491170910875873: 1, 25.464391151007327:
1, 25.425984962347698: 1, 25.37067447520296: 1, 25.119976741896828: 1,
25.045355664633238: 1, 24.98478772944157: 1, 24.890508574784377: 1, 24.
691181124937692: 1, 24.534583089462494: 1, 24.47682593783986: 1, 24.372
716310752292: 1, 24.27810710187315: 1, 24.192313191761365: 1, 24.083749
18401641: 1, 24.07636592230089: 1, 23.973725135152925: 1, 23.9658943904
47286: 1, 23.931916914352602: 1, 23.897113532020576: 1, 23.633431680289
75: 1, 23.587090646356135: 1, 23.586454571620017: 1, 23.44039571005465:
1, 23.42241158148424: 1, 23.390527289215886: 1, 23.349596088006958: 1,
23.279776220589802: 1, 23.262925520208476: 1, 23.090209896938003: 1, 2
3.002175282345902: 1, 22.921639811512602: 1, 22.84964642228734: 1, 22.7
3858250076297: 1, 22.712057963202216: 1, 22.707475557625084: 1, 22.5171
77048334613: 1, 22.49311317103321: 1, 22.490040976903586: 1, 22.3888062
65669626: 1, 22.360679588004714: 1, 22.338800121325303: 1, 22.268212892
49064: 1, 22.19951502769862: 1, 22.192805016236473: 1, 22.1522074895232
3: 1, 22.01582646078002: 1, 21.93270064488917: 1, 21.885169462508284:
1, 21.825313649243505: 1, 21.824245069360448: 1, 21.75147402450722: 1,
21.67162257967584: 1, 21.621239289804528: 1, 21.598725133220213: 1, 21.
574771569146762: 1, 21.549955480534685: 1, 21.543840508556144: 1, 21.46
8973816105002: 1, 21.361146277280874: 1, 21.344344097513247: 1, 21.3210
58728119663: 1, 21.32061146802923: 1, 21.280985440919093: 1, 21.2466678
0199311: 1, 21.222847096459407: 1, 21.200428292152672: 1, 21.1431971458
351: 1, 21.088329556599103: 1, 21.051937954693447: 1, 21.03853278176088
: 1, 21.014524491063014: 1, 20.9624209603492: 1, 20.953967650453873:
1, 20.948608233303815: 1, 20.93443508722687: 1, 20.85966896463063: 1, 2
0.810268450923072: 1, 20.78604627102906: 1, 20.71704294865541: 1, 20.69
6827763657303: 1, 20.643331935922387: 1, 20.639512989441464: 1, 20.5642
3473626471: 1, 20.55996136011081: 1, 20.377846491826872: 1, 20.30237875
264658: 1, 20.297614668222852: 1, 20.259381060762387: 1, 20.15190369209
076: 1, 20.103984379091415: 1, 20.0771121239365: 1, 20.060433272808304:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

1, 20.05055557358533: 1, 20.024895797975514: 1, 20.019693870109467: 1,
19.950735217139528: 1, 19.912279635918868: 1, 19.8953267144007: 1, 19.8
92090017604907: 1, 19.799229576730855: 1, 19.767375489059898: 1, 19.661
6398604158: 1, 19.632500211416453: 1, 19.552281920249065: 1, 19.5412847
241932: 1, 19.53892096571334: 1, 19.428838379514882: 1, 19.399329379139
36: 1, 19.386236817299512: 1, 19.330914707184768: 1, 19.32484494100019:
1, 19.31632172990843: 1, 19.306497557445113: 1, 19.27514254869852: 1, 1
9.183384121001225: 1, 19.11479357223788: 1, 19.08190501700868: 1, 19.08
0742617011197: 1, 19.075829393998795: 1, 19.036130189714623: 1, 19.0088
56574866588: 1, 18.999613797892998: 1, 18.996430774565585: 1, 18.912792
06384206: 1, 18.756346211524342: 1, 18.67454529319575: 1, 18.6737466102
8017: 1, 18.662410837071803: 1, 18.644510515984376: 1, 18.5877892478700
9: 1, 18.553189325897186: 1, 18.518488819887995: 1, 18.464901753383547:
1, 18.46329206962964: 1, 18.446956370557263: 1, 18.338725994691575: 1,
18.25094622298861: 1, 18.24211004425469: 1, 18.206129221294507: 1, 18.1
3028348786809: 1, 18.113234914515072: 1, 18.075151281366466: 1, 18.0248
90775062904: 1, 17.991682508857444: 1, 17.974711319759507: 1, 17.972722
35468998: 1, 17.97119608211791: 1, 17.935191349582137: 1, 17.9180491454
16397: 1, 17.915857374722254: 1, 17.847044536719885: 1, 17.833252727546
842: 1, 17.816983354409: 1, 17.698437435911856: 1, 17.690637361369085:
1, 17.684269577682148: 1, 17.675040018255842: 1, 17.673590498102968: 1,
17.666631131531037: 1, 17.65333022931899: 1, 17.630562137139602: 1, 17.
608781603172588: 1, 17.6065889918431: 1, 17.598154150918344: 1, 17.5221
071009524: 1, 17.50774394273972: 1, 17.454129142137493: 1, 17.453013544
00872: 1, 17.439255679512083: 1, 17.41791709605081: 1, 17.4117104572374
7: 1, 17.372044108685287: 1, 17.270009048227465: 1, 17.189664823113763:
1, 17.095219711740203: 1, 17.088937515243643: 1, 17.075878972806564: 1,
17.06197711613499: 1, 17.052049393168943: 1, 16.981651979903244: 1, 16.
97656182105501: 1, 16.954691433081287: 1, 16.936776476077803: 1, 16.922
44289286715: 1, 16.88704453285351: 1, 16.885793286389205: 1, 16.8716059
1097389: 1, 16.836477395956468: 1, 16.810318572257348: 1, 16.7627801077
21934: 1, 16.737223461331073: 1, 16.723200219612163: 1, 16.700614287891
963: 1, 16.690130449475365: 1, 16.678803102328907: 1, 16.64756692899622
3: 1, 16.620339763695448: 1, 16.61638354632296: 1, 16.603966277925984:
1, 16.59128909665567: 1, 16.553527960287038: 1, 16.508942714770527: 1,
16.49981650165112: 1, 16.4701841737509: 1, 16.411675369048243: 1, 16.39
065706640511: 1, 16.332094254366567: 1, 16.260710700872163: 1, 16.25221
388588273: 1, 16.222207053632804: 1, 16.21564421934627: 1, 16.214887619
744474: 1, 16.1610757141746: 1, 16.114154941714368: 1, 16.0265656565496

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

32: 1, 15.98935044215107: 1, 15.970532201657173: 1, 15.944707784228227:
1, 15.91760989127839: 1, 15.915640359033743: 1, 15.908869496040124: 1,
15.883576586800896: 1, 15.863835652364683: 1, 15.845937051601181: 1, 1
5.844106870432586: 1, 15.819976789807944: 1, 15.787314770180982: 1, 15.
739760155476755: 1, 15.731260686981257: 1, 15.725717082540426: 1, 15.72
2559182140605: 1, 15.71649976532839: 1, 15.705912400361083: 1, 15.69602
1548760218: 1, 15.630612159105567: 1, 15.62663790058129: 1, 15.58515879
6941494: 1, 15.54169798547567: 1, 15.537857597289598: 1, 15.44045967424
9482: 1, 15.424226803173493: 1, 15.398413085007695: 1, 15.3868294278341
54: 1, 15.381812837132049: 1, 15.371564624438468: 1, 15.3413046543866:
1, 15.172756963098886: 1, 15.159894296237516: 1, 15.15015315157226: 1,
15.128851406543511: 1, 15.12648779894864: 1, 15.10458945901174: 1, 15.1
01601942810325: 1, 15.065023084257282: 1, 15.054673673136698: 1, 14.973
825249651053: 1, 14.919936431029393: 1, 14.864329341102868: 1, 14.86427
4377143783: 1, 14.84017166686903: 1, 14.837520452508523: 1, 14.81915203
2889976: 1, 14.80531708896524: 1, 14.799382206200246: 1, 14.79050337585
5812: 1, 14.779595086123845: 1, 14.775648979523451: 1, 14.7538256512083
25: 1, 14.725178110379545: 1, 14.711882383083733: 1, 14.64501449182959
7: 1, 14.636486171421474: 1, 14.611905753365498: 1, 14.608022534500986:
1, 14.59431181547601: 1, 14.592916667476377: 1, 14.590544647795058: 1,
14.584488644110722: 1, 14.560289161010767: 1, 14.528919934504534: 1, 1
4.526126647343373: 1, 14.476060969825314: 1, 14.474609936629822: 1, 14.
416558803309329: 1, 14.412875996649396: 1, 14.411000533687295: 1, 14.39
3488645351656: 1, 14.371041029574283: 1, 14.358571161838213: 1, 14.3584
9836693263: 1, 14.324426411883572: 1, 14.319887391040648: 1, 14.3188024
48647384: 1, 14.290955275106013: 1, 14.281841628783418: 1, 14.273291169
893943: 1, 14.268232303166615: 1, 14.236832194781423: 1, 14.22194588431
8668: 1, 14.126647344351422: 1, 14.086668490175375: 1, 14.0787710414125
3: 1, 14.075575210838382: 1, 14.071076656143024: 1, 14.066573142382701:
1, 13.9992111631598: 1, 13.99607321789502: 1, 13.952101890877804: 1, 1
3.946370745593457: 1, 13.890699394901187: 1, 13.882248666881598: 1, 13.
864686366710602: 1, 13.855738561848645: 1, 13.85462602313093: 1, 13.825
818736984944: 1, 13.819250521771208: 1, 13.787088603738018: 1, 13.78266
9417986021: 1, 13.753016137301302: 1, 13.735876457837168: 1, 13.6856134
35443878: 1, 13.672451536892734: 1, 13.64344434406186: 1, 13.6135743150
4212: 1, 13.565461274745466: 1, 13.536067813309295: 1, 13.5305055914304
87: 1, 13.504605911987062: 1, 13.502999058249985: 1, 13.49544984096737
1: 1, 13.451804407311496: 1, 13.429463108302459: 1, 13.410453855674158:
1, 13.398627943273382: 1, 13.36628541769944: 1, 13.322119356166741: 1,

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

13.295430642472045: 1, 13.26991804541026: 1, 13.261539794323287: 1, 13.
208562354714717: 1, 13.198731237695197: 1, 13.096157292631066: 1, 13.07
8944207239878: 1, 13.071700763189734: 1, 13.04789234971013: 1, 13.03576
5093752863: 1, 13.03150829925239: 1, 13.022975875355671: 1, 13.01947683
7727087: 1, 13.013921731757724: 1, 13.013346247742353: 1, 12.9866291197
61913: 1, 12.978921576971617: 1, 12.943329860172296: 1, 12.897227941606
193: 1, 12.892258385629406: 1, 12.842017986910301: 1, 12.82935741372745
2: 1, 12.821780164664414: 1, 12.792407716842735: 1, 12.753860667646174:
1, 12.74774356176144: 1, 12.711730233275985: 1, 12.706123172259904: 1,
12.679063320135134: 1, 12.676894980093588: 1, 12.668600117467129: 1, 1
2.592470614911509: 1, 12.579309390183584: 1, 12.560466005005967: 1, 12.
557679882297455: 1, 12.5388155972752: 1, 12.5001950912772: 1, 12.494827
4832529: 1, 12.47716904638377: 1, 12.474866590475319: 1, 12.45724915466
2575: 1, 12.443279780328746: 1, 12.435755893765881: 1, 12.3884839803684
6: 1, 12.38604434839254: 1, 12.381059703718556: 1, 12.357831890799915:
1, 12.351211827447896: 1, 12.332147076679737: 1, 12.307631031112122: 1,
12.304180779276551: 1, 12.300339705261784: 1, 12.26815324490863: 1, 12.
241121345845839: 1, 12.226962770122434: 1, 12.196009648699315: 1, 12.19
3454667030815: 1, 12.190696316485834: 1, 12.173826562342175: 1, 12.1538
09311014827: 1, 12.153382161025263: 1, 12.151384287204934: 1, 12.146434
438077277: 1, 12.136135789454942: 1, 12.13455970419545: 1, 12.076835392
06991: 1, 12.073753093943342: 1, 12.050634856007022: 1, 12.046459581569
23: 1, 12.036644952338833: 1, 12.001540441137253: 1, 11.99835284704999
7: 1, 11.978254633529845: 1, 11.970565893482048: 1, 11.957323486988845:
1, 11.945220715539907: 1, 11.917562597544894: 1, 11.914510785680996: 1,
11.899451107342477: 1, 11.894870527739739: 1, 11.884941233870443: 1, 1
1.87959025799626: 1, 11.871479090006988: 1, 11.83813458342683: 1, 11.83
7542154307249: 1, 11.82109606858991: 1, 11.81728005795754: 1, 11.798804
925013226: 1, 11.793882938924328: 1, 11.787453941318102: 1, 11.78080003
418621: 1, 11.78044972201858: 1, 11.763846454714065: 1, 11.757079505803
077: 1, 11.739977026537133: 1, 11.73743895259961: 1, 11.71721510089498
6: 1, 11.711045796776936: 1, 11.693988663336217: 1, 11.660610489195262:
1, 11.660255744311826: 1, 11.643637990133408: 1, 11.630144499500238: 1,
11.603854449506548: 1, 11.59409596960553: 1, 11.593539194895374: 1, 11.
568552337223394: 1, 11.566365958887904: 1, 11.516517598455648: 1, 11.50
7109991249305: 1, 11.448576528159101: 1, 11.425793829499797: 1, 11.4154
5682916524: 1, 11.4086321098462: 1, 11.394366812669151: 1, 11.375947254
094381: 1, 11.371593807314063: 1, 11.345412112095696: 1, 11.34469479056
5514: 1, 11.316588505861839: 1, 11.296075915767457: 1, 11.265357217004

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
1: 1, 11.260910351242824: 1, 11.222921794674972: 1, 11.220932132012113:
1, 11.2191193764906: 1, 11.21565698980988: 1, 11.206264656293596: 1, 1
1.204770719327716: 1, 11.178972060615337: 1, 11.167627135928582: 1, 11.
12732247945369: 1, 11.09566078916361: 1, 11.08759329213583: 1, 11.07781
7863858433: 1, 11.062314931918312: 1, 11.055778085832141: 1, 11.0387321
28637722: 1, 11.032966314749226: 1, 11.023589654924887: 1, 11.020344001
170434: 1, 11.017533413632531: 1, 11.016195619172445: 1, 10.99876283343
1465: 1, 10.992214380065684: 1, 10.961416907556101: 1, 10.9427444748385
43: 1, 10.93878686842539: 1, 10.93447383562521: 1, 10.934158707700995:
1, 10.916512163462174: 1, 10.913067410505748: 1, 10.85936801415533: 1,
10.8417866512641: 1, 10.8335833435276: 1, 10.815086864560197: 1, 10.805
282473377694: 1, 10.790145081967236: 1, 10.769350289169784: 1, 10.76626
061762587: 1, 10.755684751799935: 1, 10.753862223021752: 1, 10.74816496
9293516: 1, 10.724001628913888: 1, 10.713019578020047: 1, 10.7120594152
46017: 1, 10.709760656924432: 1, 10.702763796339912: 1, 10.697915735052
256: 1, 10.694493062260971: 1, 10.687477355533021: 1, 10.67862614278966
4: 1, 10.649133188198663: 1, 10.600090552298406: 1, 10.570340382840719:
1, 10.562929261915201: 1, 10.551759014860636: 1, 10.54100906508347: 1,
10.533324384659055: 1, 10.53050129668297: 1, 10.530335314966198: 1, 10.
510483011308146: 1, 10.49931366440117: 1, 10.492367868703472: 1, 10.485
940191139228: 1, 10.47899066194109: 1, 10.475551164697507: 1, 10.474327
131787346: 1, 10.442441315288526: 1, 10.42199881924373: 1, 10.418130705
136347: 1, 10.411652384586045: 1, 10.381093914264744: 1, 10.36644928329
866: 1, 10.344118822389369: 1, 10.342635766918846: 1, 10.3359443695155
8: 1, 10.311342221705226: 1, 10.30208636717727: 1, 10.282033977491338:
1, 10.275297729107653: 1, 10.269004162037202: 1, 10.265057303993459: 1,
10.254907302022445: 1, 10.203259401202931: 1, 10.197208056183328: 1, 1
0.189453037406683: 1, 10.182961772266662: 1, 10.163059235154629: 1, 10.
16223493606808: 1, 10.156942337432993: 1, 10.15325323180067: 1, 10.1479
80738496205: 1, 10.138165577428039: 1, 10.11909834764576: 1, 10.0938815
70497123: 1, 10.08741303790172: 1, 10.0870620561994: 1, 10.076206513797
409: 1, 10.065930090004052: 1, 10.060539525291665: 1, 10.05565514805993
2: 1, 10.050014568196863: 1, 10.036437656185988: 1, 10.035028853042183:
1, 10.03047868163196: 1, 10.012945868690618: 1, 10.008806824825053: 1,
9.989761526734354: 1, 9.987421451357916: 1, 9.98662251409011: 1, 9.9855
3891105073: 1, 9.980514927013058: 1, 9.963032925167962: 1, 9.9626530349
0606: 1, 9.939307996476513: 1, 9.92837238908265: 1, 9.910898119608186:
1, 9.892257497497662: 1, 9.88820683777158: 1, 9.860893020416176: 1, 9.8
60577399058668: 1, 9.836194134558063: 1, 9.83166636532569: 1, 9.8033956
```


Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

65794458: 1, 9.802013946836128: 1, 9.793388274723613: 1, 9.793173036283
365: 1, 9.758768722099731: 1, 9.756656695059572: 1, 9.751787839023644:
1, 9.74567166766161: 1, 9.740285590019539: 1, 9.730493925807913: 1, 9.7
24382743635038: 1, 9.722664336462033: 1, 9.71087246103959: 1, 9.7030406
65022943: 1, 9.687164756461884: 1, 9.669720384766917: 1, 9.651302131944
435: 1, 9.63781270855165: 1, 9.630151125944053: 1, 9.623901658640047:
1, 9.612716912134568: 1, 9.596958636104244: 1, 9.595483170495683: 1, 9.
58569755562501: 1, 9.56585713699887: 1, 9.565665356817865: 1, 9.5619326
67360784: 1, 9.56192607348256: 1, 9.551550950391423: 1, 9.4955184182154
58: 1, 9.453238384345685: 1, 9.45312825269401: 1, 9.446933385579365: 1,
9.436963024393918: 1, 9.427411092447862: 1, 9.422363622466218: 1, 9.407
290454089225: 1, 9.394707324773398: 1, 9.392607089150104: 1, 9.38536425
3429438: 1, 9.375772338930714: 1, 9.37377531858445: 1, 9.37168132854939
5: 1, 9.353388825760668: 1, 9.347213740811227: 1, 9.345482506804526: 1,
9.34416836239116: 1, 9.342034597053091: 1, 9.330992937837552: 1, 9.3271
51173169993: 1, 9.29420227169282: 1, 9.280855991669089: 1, 9.2709105626
25316: 1, 9.261484687886615: 1, 9.252381501619473: 1, 9.24192740619797
2: 1, 9.234374016332207: 1, 9.233820069572051: 1, 9.229807967594025: 1,
9.217213122115131: 1, 9.211283262978085: 1, 9.19806809534249: 1, 9.1952
3256799338: 1, 9.179245086131074: 1, 9.174193474645694: 1, 9.1708867342
59136: 1, 9.170337648166003: 1, 9.170180430522805: 1, 9.16802811160671
6: 1, 9.15886334203037: 1, 9.156732683770416: 1, 9.155563281095292: 1,
9.134395326449322: 1, 9.119830049198615: 1, 9.113220816653955: 1, 9.110
361536908076: 1, 9.106296190889415: 1, 9.095408901106117: 1, 9.09412973
0867614: 1, 9.093270717228485: 1, 9.092817004244564: 1, 9.0828872195653
96: 1, 9.074296950308726: 1, 9.065109400738898: 1, 9.059787944151175:
1, 9.056089227370883: 1, 9.05245441127544: 1, 9.050949921106245: 1, 9.0
35849466105041: 1, 9.032598342620183: 1, 9.030232706773965: 1, 8.986999
759484311: 1, 8.983101884794122: 1, 8.978891070605082: 1, 8.97857774933
104: 1, 8.953120436098478: 1, 8.951884296344208: 1, 8.950999086333521:
1, 8.920961622555868: 1, 8.915010453308513: 1, 8.910820594253897: 1, 8.
898203325617377: 1, 8.896484657919013: 1, 8.896104973573289: 1, 8.89474
0356299168: 1, 8.863380127510638: 1, 8.863201658206535: 1, 8.8575650285
00863: 1, 8.853509628912152: 1, 8.853245844116017: 1, 8.82536512874123
8: 1, 8.805808982185399: 1, 8.80170282520832: 1, 8.785726694289224: 1,
8.782644056389035: 1, 8.776853567744018: 1, 8.74865722209855: 1, 8.7330
90822019934: 1, 8.731637331415723: 1, 8.728153340013744: 1, 8.724563386
400169: 1, 8.715126330239551: 1, 8.711878742397799: 1, 8.70055685973422
6: 1, 8.683776890562905: 1, 8.658254436235653: 1, 8.632058486575561: 1,

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

8.631264401478166: 1, 8.629487676363912: 1, 8.599958575842502: 1, 8.596
317916187473: 1, 8.578900150733672: 1, 8.571418484067706: 1, 8.57069081
4510433: 1, 8.564896291200707: 1, 8.517580692694622: 1, 8.5096273104796
12: 1, 8.493847760041465: 1, 8.479745977903383: 1, 8.477979690880256:
1, 8.462381518342172: 1, 8.434706358435687: 1, 8.434455606008635: 1, 8.
42562136685997: 1, 8.419857109210644: 1, 8.40347794158775: 1, 8.3783266
37690442: 1, 8.365852347639398: 1, 8.358944658937753: 1, 8.344640416965
403: 1, 8.343155219057577: 1, 8.327976327860638: 1, 8.321003665236418:
1, 8.316738287774278: 1, 8.308066427679396: 1, 8.284064031853502: 1, 8.
264540636850418: 1, 8.252209320539023: 1, 8.250113161378502: 1, 8.23562
2156508088: 1, 8.222852865330708: 1, 8.17644093167786: 1, 8.16805260621
8936: 1, 8.145134438506817: 1, 8.140941989512964: 1, 8.140760209492353:
1, 8.140351942011586: 1, 8.140022753757314: 1, 8.129089861493453: 1, 8.
119744753543985: 1, 8.094054345545883: 1, 8.076982667764229: 1, 8.07047
0006731169: 1, 8.068438866362737: 1, 8.065200888294282: 1, 8.0505333780
65895: 1, 8.030491162457139: 1, 8.027066978860782: 1, 8.02118367670288
6: 1, 8.016092763402305: 1, 8.015937994340733: 1, 8.014071893354599: 1,
7.989679772169171: 1, 7.9821489800789225: 1, 7.95244861124952: 1, 7.952
134433575156: 1, 7.948831130323015: 1, 7.920436782319197: 1, 7.90475659
0383331: 1, 7.8884147340266475: 1, 7.882149220545374: 1, 7.873426804295
209: 1, 7.8671464507106865: 1, 7.855244110378925: 1, 7.853713870222919
4: 1, 7.852401352151161: 1, 7.839112305583612: 1, 7.835900018053659: 1,
7.79349829824894: 1, 7.776450977326568: 1, 7.771028984847414: 1, 7.7408
57173019775: 1, 7.733902717852519: 1, 7.7255993014250555: 1, 7.72173896
2897311: 1, 7.718189346315314: 1, 7.70230467859833: 1, 7.7020229468502
4: 1, 7.669335124426867: 1, 7.6564203391781565: 1, 7.63345854026258: 1,
7.614098288793155: 1, 7.600674841387025: 1, 7.59532406037913: 1, 7.5822
87124507191: 1, 7.581058340426612: 1, 7.572498357141357: 1, 7.560348062
133681: 1, 7.55773695958015: 1, 7.555166932349001: 1, 7.48942806830900
7: 1, 7.4803654332906095: 1, 7.475990216619719: 1, 7.455083112063328:
1, 7.4430810911671825: 1, 7.44196966295187: 1, 7.439660474718081: 1, 7.
424485612941275: 1, 7.4223651096053915: 1, 7.409609709919526: 1, 7.3979
79748821753: 1, 7.392461218193557: 1, 7.371683118228944: 1, 7.345196880
492518: 1, 7.339643112802949: 1, 7.3231518957496515: 1, 7.2791924262090
2: 1, 7.242537672786742: 1, 7.22651002815225: 1, 7.215536838608005: 1,
7.2141668569892134: 1, 7.1948382733123815: 1, 7.181501015122363: 1, 7.1
68147293129896: 1, 7.166741061097946: 1, 7.14440387788482: 1, 7.1279106
08993791: 1, 7.118964901099679: 1, 7.106525212099789: 1, 6.996012030820
375: 1, 6.973403631824012: 1, 6.953140259713686: 1, 6.94293932537601:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
1, 6.940554712630928: 1, 6.922289284435831: 1, 6.91704039529053: 1, 6.8
888579713461375: 1, 6.860785994812926: 1, 6.849158155233598: 1, 6.80687
070531856: 1, 6.719974772586151: 1, 6.7072138237724594: 1, 6.5884386313
38921: 1, 6.509975324378893: 1, 6.472607450311211: 1, 6.46001116403065
3: 1})
```

In [45]:

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

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

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
```

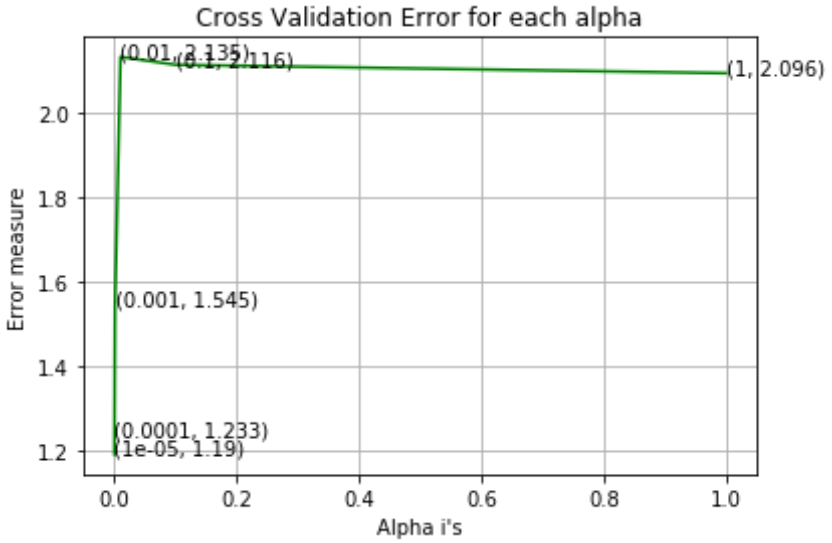
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1898797086405262
For values of alpha =  0.0001 The log loss is: 1.2328299211067009
For values of alpha =  0.001 The log loss is: 1.5449524376587334
For values of alpha =  0.01 The log loss is: 2.1345990252496
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
For values of alpha =  0.1 The log loss is: 2.115777630171907
For values of alpha =  1 The log loss is: 2.095768457322865
```




```
For values of best alpha =  1e-05 The train log loss is: 0.780029230981
3975
For values of best alpha =  1e-05 The cross validation log loss is: 1.1
898797086405262
For values of best alpha =  1e-05 The test log loss is: 1.1808992643858
385
```

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

Ans. Yes, it seems like!

In [50]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

In [51]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
 train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appe
ared in train data")
```

```
3.444 % of word of test data appeared in train data
3.894 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

First let's do some feature engineering

Adding genes and variation

In [52]:
```
train_df['gene_var']=train_df['Gene']+' '+train_df['Variation']
test_df['gene_var']=test_df['Gene']+' '+test_df['Variation']
cv_df['gene_var']=cv_df['Gene']+' '+cv_df['Variation']
train_df.head()
```

Out[52]:

	ID	Gene	Variation	Class	TEXT	gene_var
3113	3113	RAD51C	G264S	3	strong evidence overtly inactivating mutations...	RAD51C G264S

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	ID	Gene	Variation	Class	TEXT	gene_var
1672	1672	FLT3	D835E	7	mutations receptor tyrosine kinases implicated...	FLT3 D835E
2523	2523	BRCA1	L1854P	1	genetic screening breast ovarian cancer suscep...	BRCA1 L1854P
298	298	CHEK2	E321K	4	maintenance genomic integrity depends coordina...	CHEK2 E321K
2729	2729	BRAF	K601Q	7	noonan leopard cardiofaciocutaneous syndromes ...	BRAF K601Q

Adding length of text as other feature

In [53]:
```
train_df['length']=train_df['TEXT'].apply(lambda x: len(x.split()))
```

In [54]:
```
test_df['length']=test_df['TEXT'].apply(lambda x: len(x.split()))
```

In [55]:
```
cv_df['length']=cv_df['TEXT'].apply(lambda x: len(x.split()))
```

In [56]:
```
train_df.head()
```

Out[56]:

	ID	Gene	Variation	Class	TEXT	gene_var	length
3113	3113	RAD51C	G264S	3	strong evidence overtly inactivating mutations...	RAD51C G264S	11517
1672	1672	FLT3	D835E	7	mutations receptor tyrosine kinases implicated...	FLT3 D835E	7746
2523	2523	BRCA1	L1854P	1	genetic screening breast ovarian cancer suscep...	BRCA1 L1854P	3538
298	298	CHEK2	E321K	4	maintenance genomic integrity depends coordina...	CHEK2 E321K	2649
2729	2729	BRAF	K601Q	7	noonan leopard cardiofaciocutaneous syndromes ...	BRAF K601Q	3351

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [57]:
```
#performing tfidf on new_feature
vectorizer=TfidfVectorizer(min_df=3)
genevar_feature_train_tfidf=vectorizer.fit_transform(train_df['gene_va
r'])
genevar_feature_test_tfidf=vectorizer.transform(test_df['gene_var'])
genevar_feature_cv_tfidf=vectorizer.transform(cv_df['gene_var'])
genevar_feature_test_tfidf.shape
```

Out[57]:
```
(665, 149)
```

In [58]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilit
ies belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [59]:
```
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 [60]:
```
# this function will be used just for naive bayes
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
 or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point
 [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data p
oint [{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point
 [{}]".format(word,yes_no))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

Stacking the three types of features

In [107]:

```
# 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_tfidf = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_tfidf))
test_gene_var_tfidf = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_tfidf))
cv_gene_var_tfidf = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_tfidf))

train_x_tfidf = hstack((train_gene_var_tfidf, train_text_feature_tfidf
))
train_x_tfidf=hstack((train_x_tfidf,genevar_feature_train_tfidf)).tocsr
()

train_y = np.array(list(train_df['Class']))

test_x_tfidf = hstack((test_gene_var_tfidf, test_text_feature_tfidf))
test_x_tfidf=hstack((test_x_tfidf,genevar_feature_test_tfidf)).tocsr()

test_y = np.array(list(test_df['Class']))

cv_x_tfidf = hstack((cv_gene_var_tfidf, cv_text_feature_tfidf))
cv_x_tfidf=hstack((cv_x_tfidf,genevar_feature_cv_tfidf)).tocsr()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
cv_y = np.array(list(cv_df['Class']))


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

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

In [108]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", t
est_x_tfidf.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_tfidf.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 33
26)
(number of data points * number of features) in test data =  (665, 332
6)
(number of data points * number of features) in cross validation data =
(532, 3326)
```

In [109]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_x_responseCoding.shape)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print("(number of data points * number of features) in cross validation
 data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data =
(532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [110]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# ----------------------


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

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
```

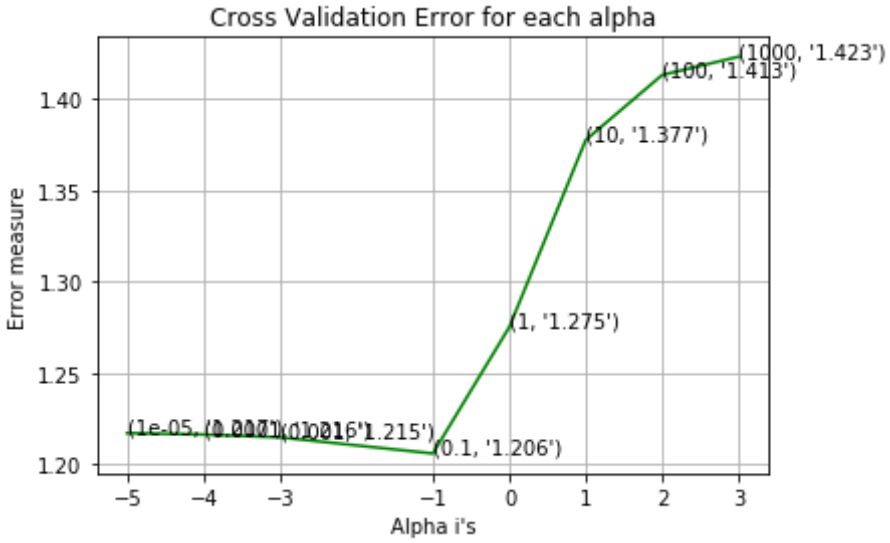
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)


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

```
for alpha = 1e-05
Log Loss : 1.2171292377044796
for alpha = 0.0001
Log Loss : 1.2163680318296042
for alpha = 0.001
Log Loss : 1.214850155304099
for alpha = 0.1
Log Loss : 1.2059994499826026
for alpha = 1
Log Loss : 1.2751299111227385
for alpha = 10
Log Loss : 1.3774029996700592
for alpha = 100
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Log Loss : 1.4129307138412428
for alpha = 1000
Log Loss : 1.4229202045000005
```




```
For values of best alpha =  0.1 The train log loss is: 0.737800902075
8673
For values of best alpha =  0.1 The cross validation log loss is: 1.2
059994499826026
For values of best alpha =  0.1 The test log loss is: 1.2537512299529
092
```

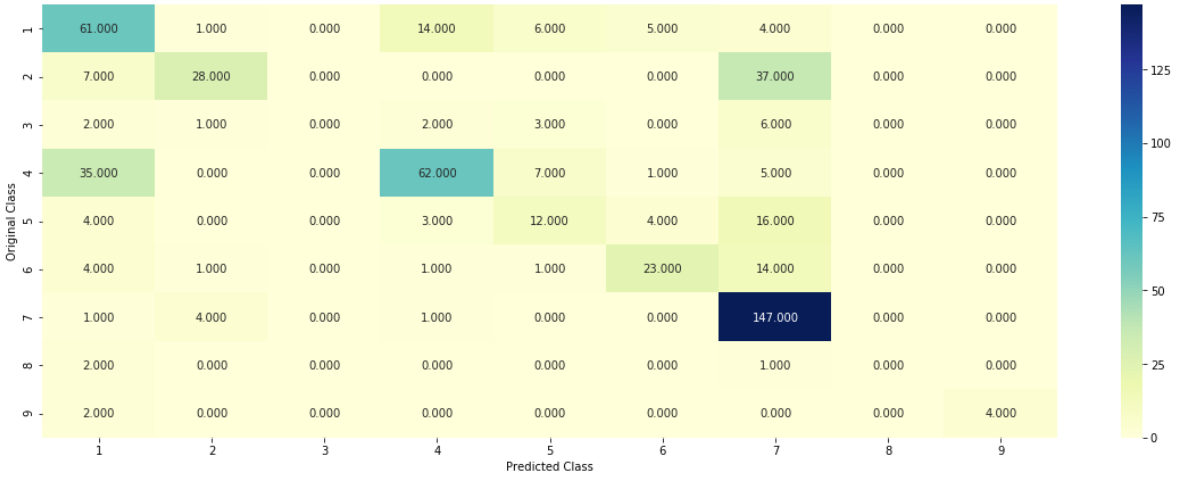
In [181]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

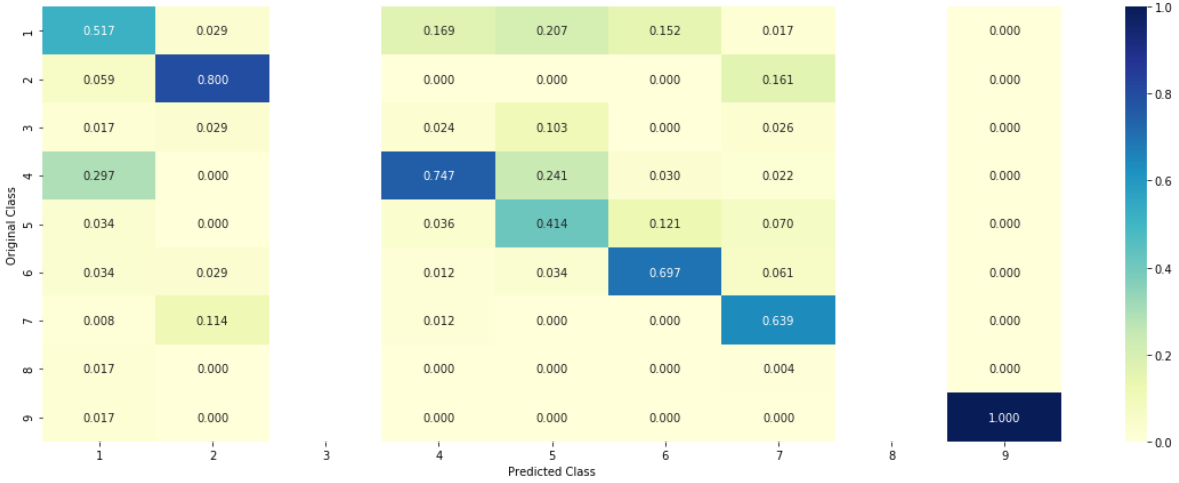
```
# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to
 X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test v
ector X.
# ----------------------
# video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


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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Log Loss : 1.1977870213780544
Number of missclassified point : 0.36654135338345867
-------------------- Confusion matrix --------------------
```

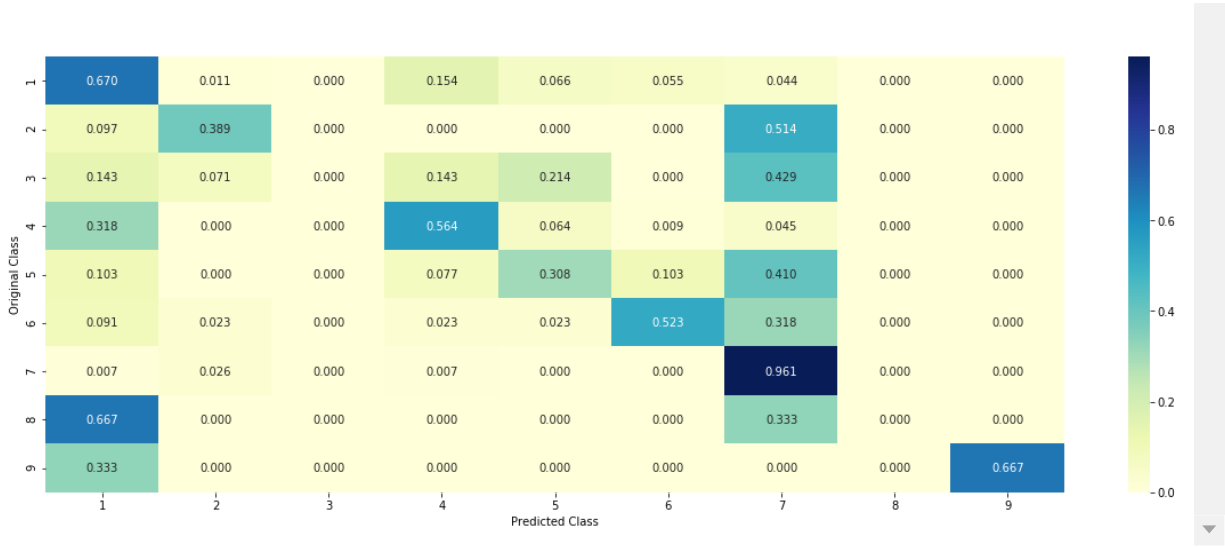



```
-------------------- Precision matrix (Columm Sum=1) ----------
----
```




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



4.1.1.3. Feature Importance, Correctly classified point

In [182]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0626 0.2167 0.018  0.0728 0.042  0.0
364 0.5422 0.005  0.0043]]
Actual Class : 7
--------------------------------------------------
19 Text feature [136] present in test data point [True]
41 Text feature [103] present in test data point [True]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
45 Text feature [1000] present in test data point [True]
59 Text feature [121] present in test data point [True]
62 Text feature [117] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [183]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.1817 0.4698 0.0197 0.0858 0.0464 0.0
64  0.1223 0.0055 0.0047]]
Actual Class : 1
--------------------------------------------------
64 Text feature [126] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [184]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

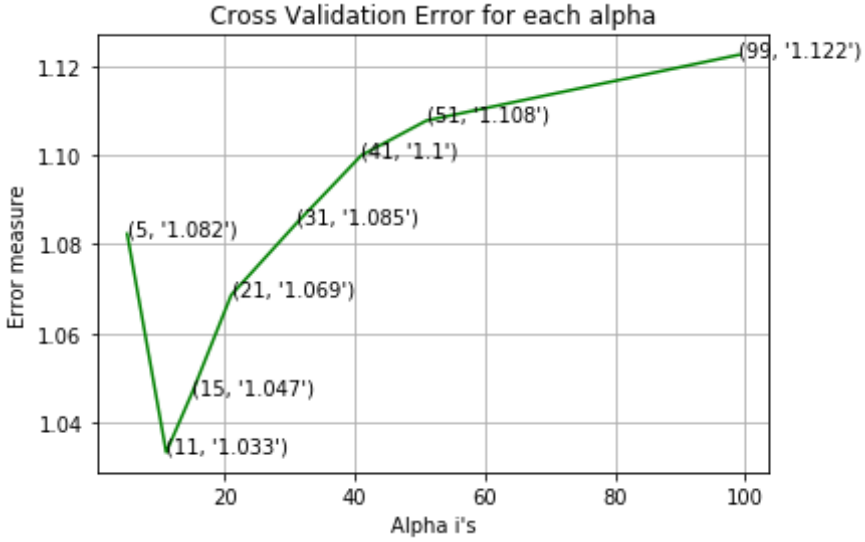
```
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
for alpha = 5
Log Loss : 1.0822975062307711
for alpha = 11
Log Loss : 1.0332769626225204
for alpha = 15

Log Loss : 1.046702284141928
for alpha = 21
Log Loss : 1.0685954592362141
for alpha = 31
Log Loss : 1.0846317570028157
for alpha = 41
Log Loss : 1.099910066073671
for alpha = 51
Log Loss : 1.1077079766830094
for alpha = 99
Log Loss : 1.1224635700858498
```




```
For values of best alpha =  11 The train log loss is: 0.638153964457770
4
For values of best alpha =  11 The cross validation log loss is: 1.0332
769626225204
For values of best alpha =  11 The test log loss is: 1.0375018906497508
```

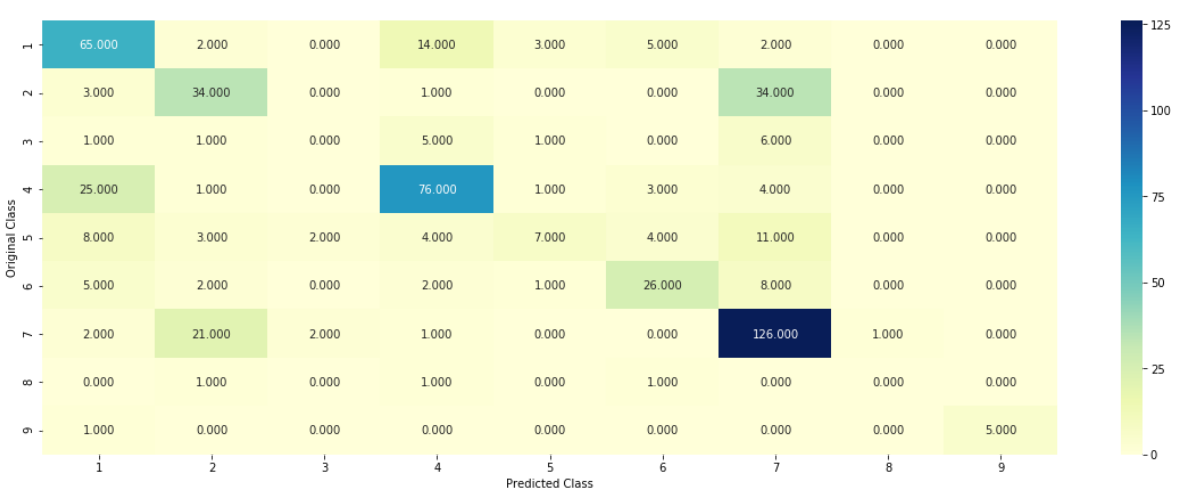
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.2.2. Testing the model with best hyper paramters

In [185]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.0332769626225204
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```

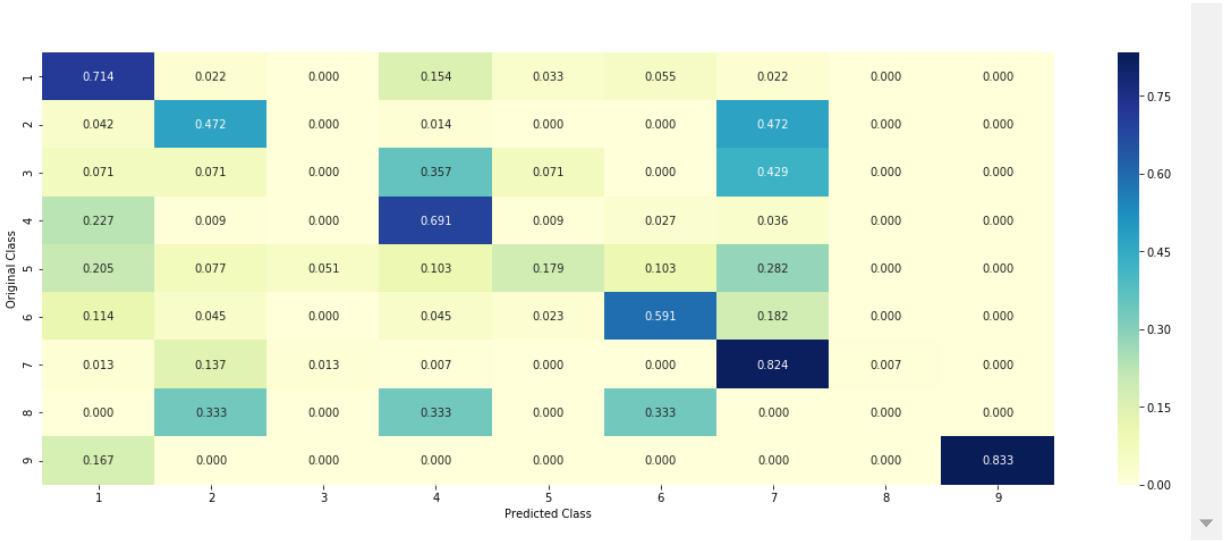
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



4.2.3.Sample Query point -1

In [186]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 4
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
2 7 7 7 7 7 7 2 7]
Fequency of nearest points : Counter({7: 9, 2: 2})
```

4.2.4. Sample Query Point-2

In [187]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

```
Predicted Class : 2
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test points
belongs to classes [2 2 1 8 4 6 4 2 2 6 5]
Fequency of nearest points : Counter({2: 4, 4: 2, 6: 2, 1: 1, 8: 1, 5:
1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [188]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

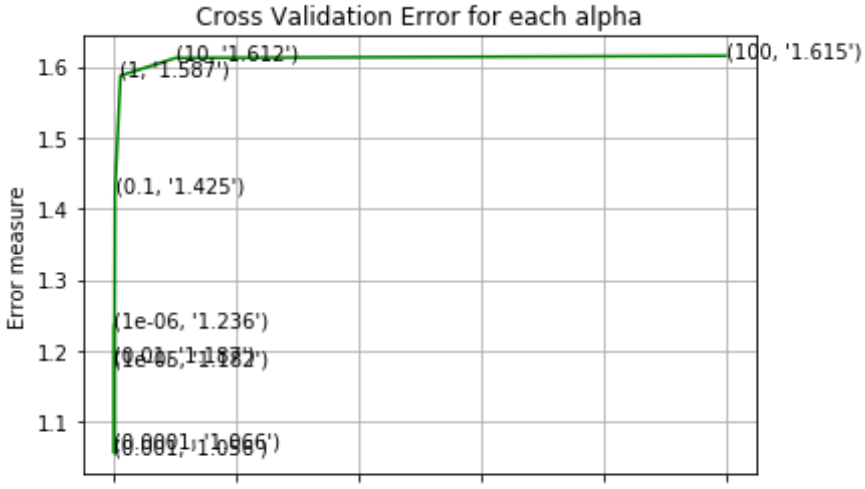
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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


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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
for alpha = 1e-06
Log Loss : 1.2357632203964266
for alpha = 1e-05
Log Loss : 1.1817195872143726
for alpha = 0.0001
Log Loss : 1.0657087344872798
for alpha = 0.001
Log Loss : 1.0557777980686025
for alpha = 0.01
Log Loss : 1.1868078319135047
for alpha = 0.1
Log Loss : 1.4250002693466879
for alpha = 1
Log Loss : 1.5871143122449194
for alpha = 10
Log Loss : 1.6120245076375779
for alpha = 100
Log Loss : 1.6150153439247974
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD


```
For values of best alpha =  0.001 The train log loss is: 0.730394216372
4654

For values of best alpha =  0.001 The cross validation log loss is: 1.0
557777980686025
For values of best alpha =  0.001 The test log loss is: 1.0246509687898
993
```

In [189]:

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

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

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

```
Log loss : 1.0557777980686025
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.3.1.3. Feature Importance

In [191]:
```
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_tfidf.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
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    print(word_present, "most importent features are present in our que
ry point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

4.3.1.3.1. Correctly Classified point

In [192]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[7.300e-03 2.348e-01 2.700e-03 8.300e-0
3 1.220e-02 4.300e-03 7.279e-01
  2.100e-03 4.000e-04]]
Actual Class : 7
--------------------------------------------------
0 Text feature [136] present in test data point [True]
128 Text feature [106] present in test data point [True]
139 Text feature [121] present in test data point [True]
160 Text feature [119] present in test data point [True]
168 Text feature [118] present in test data point [True]
212 Text feature [103] present in test data point [True]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
212 Text feature [165] present in test data point [True]
347 Text feature [1000] present in test data point [True]
432 Text feature [07] present in test data point [True]
497 Text feature [050] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [193]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.1828 0.5343 0.0023 0.0964 0.0353 0.0
367 0.108  0.0035 0.0006]]
Actual Class : 1
--------------------------------------------------
21 Text feature [132] present in test data point [True]
232 Text feature [113] present in test data point [True]
463 Text feature [14] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [194]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

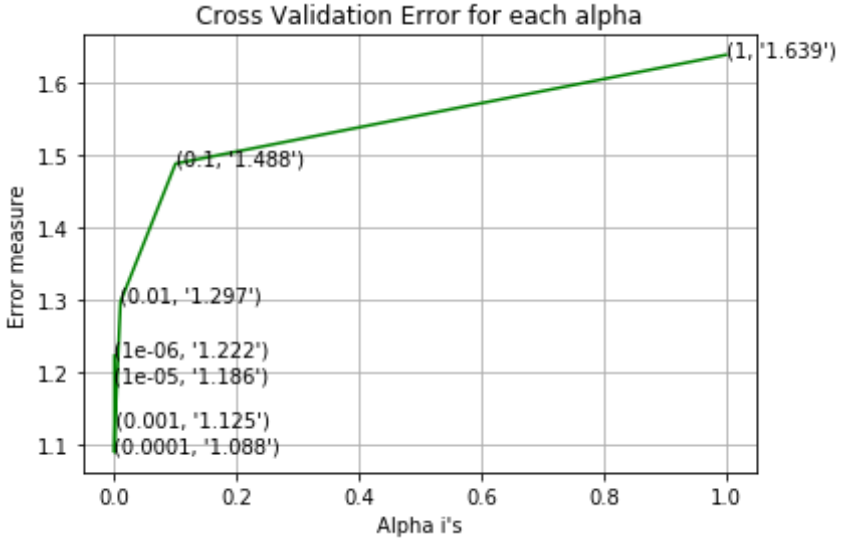
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
for alpha = 1e-06

Log Loss : 1.2221352862872188
for alpha = 1e-05
Log Loss : 1.1861968055785281
for alpha = 0.0001
Log Loss : 1.0883205483633918
for alpha = 0.001
Log Loss : 1.1248041981708472
for alpha = 0.01
Log Loss : 1.2967011494670055
for alpha = 0.1
Log Loss : 1.4879881643017183
for alpha = 1
Log Loss : 1.6386393416209106
```




```
For values of best alpha =  0.0001 The train log loss is: 0.45379890170
239007
For values of best alpha =  0.0001 The cross validation log loss is: 1.
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0883205483633918
For values of best alpha =  0.0001 The test log loss is: 1.033941102093
9604
```

4.3.2.2. Testing model with best hyper parameters

In [195]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, c
v_y, clf)
```

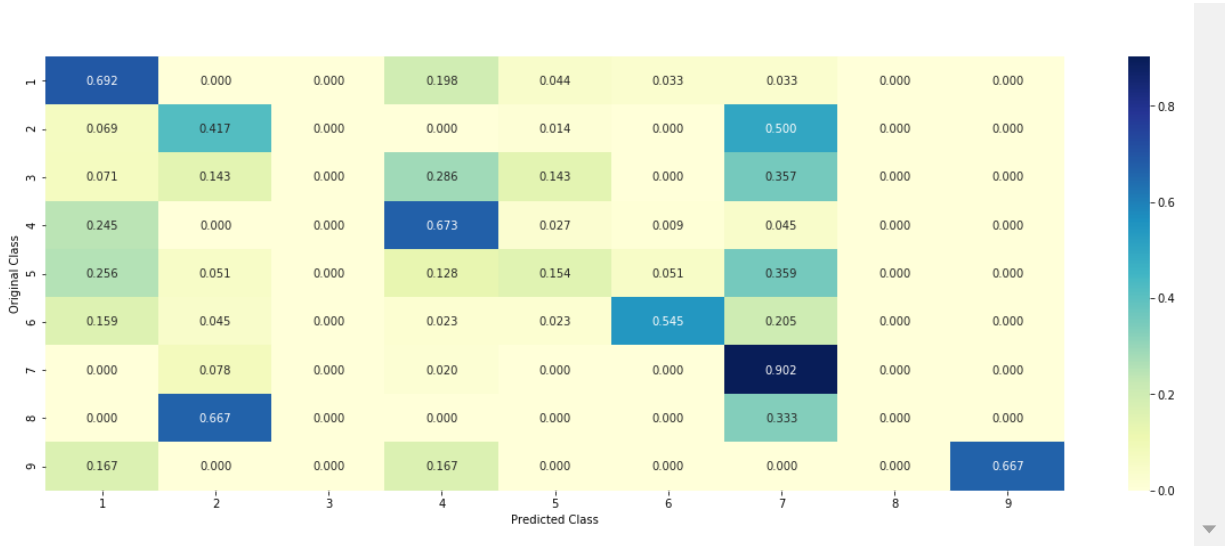
```
Log loss : 1.0883205483633918
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



4.3.2.3. Feature Importance, Correctly Classified point

In [196]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.320e-02 2.494e-01 1.100e-03 1.640e-0
2 1.150e-02 4.900e-03 7.025e-01
  7.000e-04 1.000e-04]]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Actual Class : 7
--------------------------------------------------
45 Text feature [136] present in test data point [True]
132 Text feature [121] present in test data point [True]
277 Text feature [131] present in test data point [True]
278 Text feature [106] present in test data point [True]
380 Text feature [1014] present in test data point [True]
415 Text feature [119] present in test data point [True]
433 Text feature [015] present in test data point [True]
488 Text feature [006] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [197]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[9.190e-02 7.008e-01 2.000e-04 4.800e-0
2 8.400e-03 1.730e-02 1.331e-01
  4.000e-04 0.000e+00]]
Actual Class : 1
--------------------------------------------------
177 Text feature [132] present in test data point [True]
296 Text feature [113] present in test data point [True]
356 Text feature [14] present in test data point [True]
387 Text feature [100] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [198]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

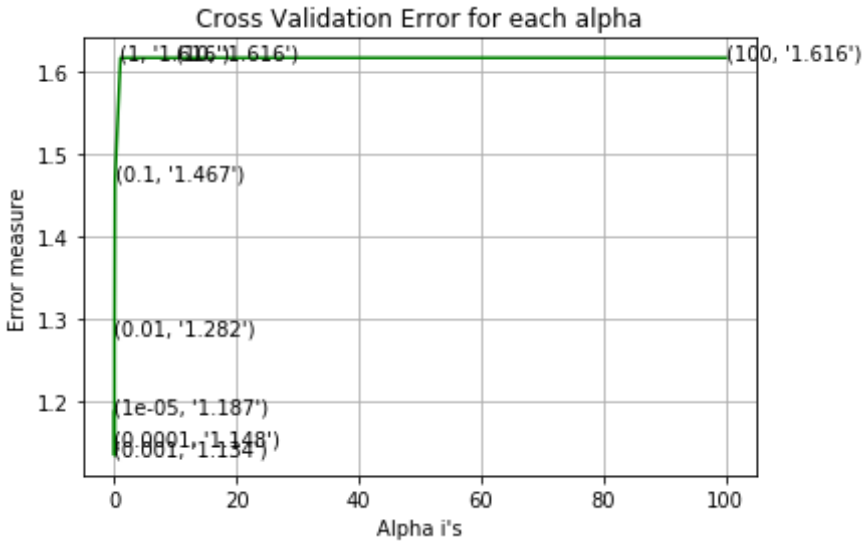
```
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='hinge', random_state=42)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
for C = 1e-05
Log Loss : 1.1871809792531485
for C = 0.0001
Log Loss : 1.1475831906656149
for C = 0.001
Log Loss : 1.1342620889801138
for C = 0.01
Log Loss : 1.281541483807246
for C = 0.1
Log Loss : 1.4674675174814182
for C = 1
Log Loss : 1.6158017893162016
for C = 10
Log Loss : 1.6157911572665105
for C = 100
Log Loss : 1.6157911549590183
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
For values of best alpha =  0.001 The train log loss is: 0.6639333854
572491
For values of best alpha =  0.001 The cross validation log loss is:
1.1342620889801138
For values of best alpha =  0.001 The test log loss is: 1.10058466435
6535
```

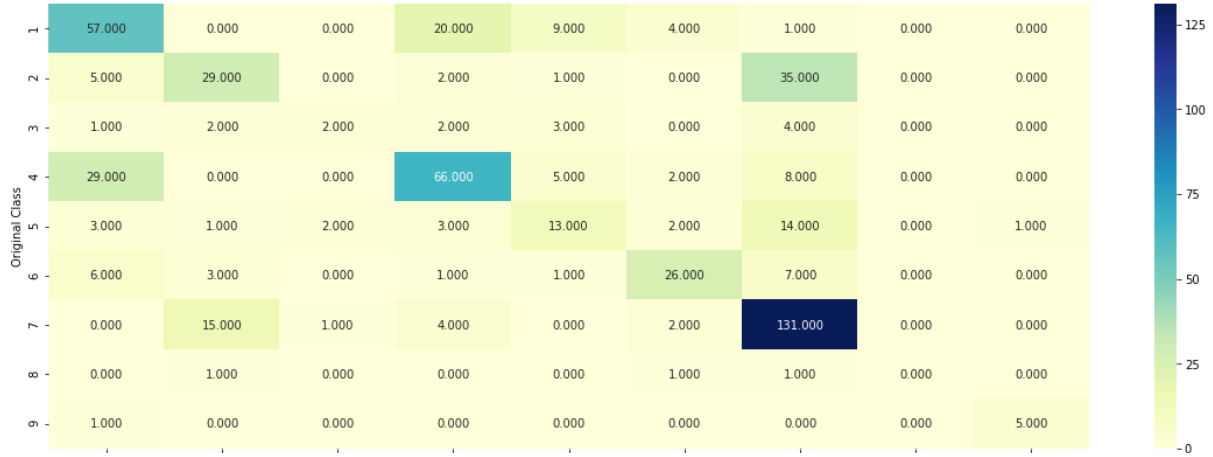
4.4.2. Testing model with best hyper parameters

In [199]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class
_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_
y, clf)
```

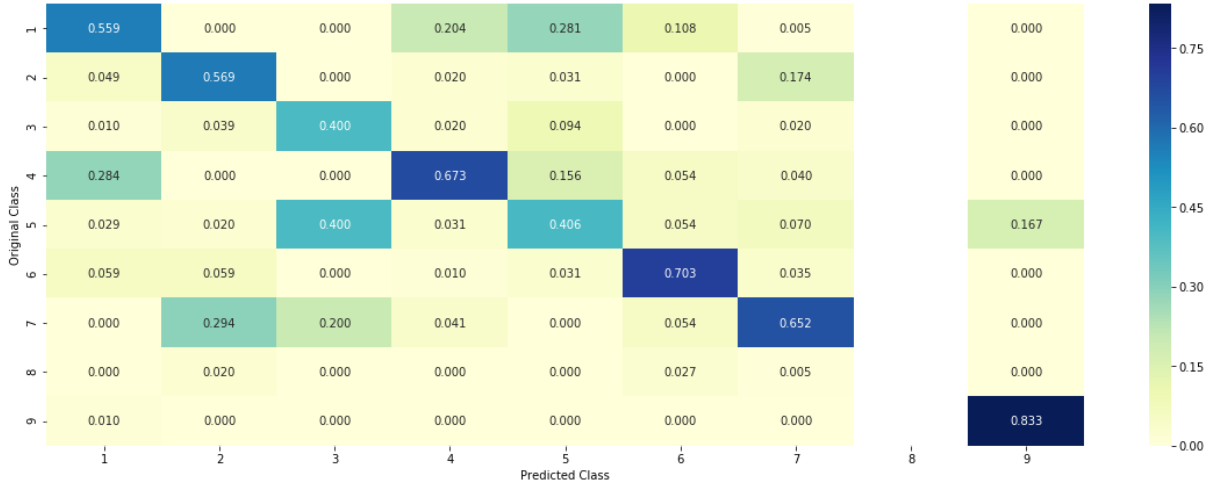
```
Log loss : 1.1342620889801138
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```



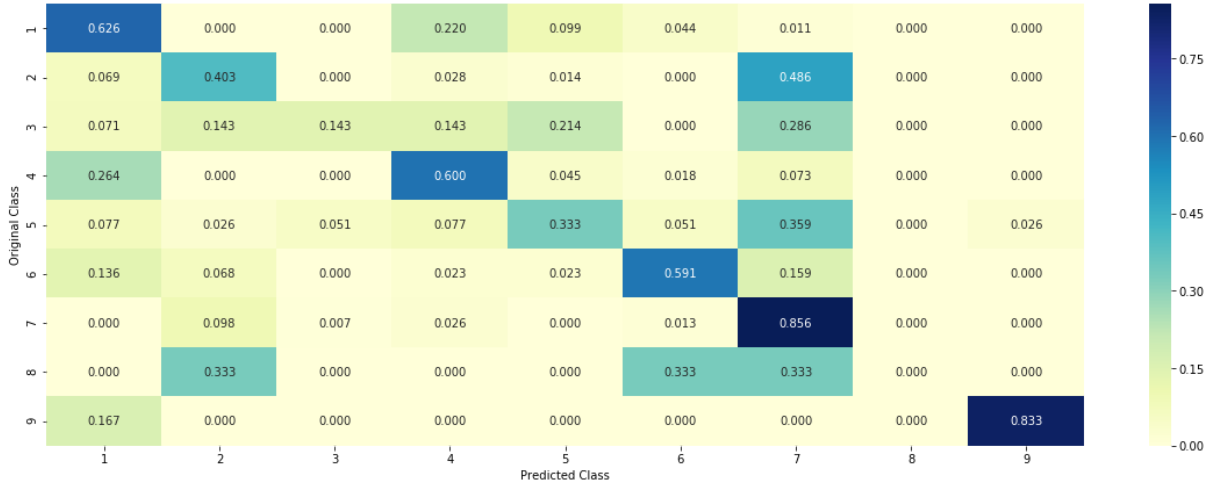
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

--



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



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [200]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0788 0.2544 0.0044 0.0601 0.0541 0.0
277 0.5167 0.0016 0.0022]]
Actual Class : 7
--------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
13 Text feature [136] present in test data point [True]
63 Text feature [121] present in test data point [True]
195 Text feature [106] present in test data point [True]
366 Text feature [1014] present in test data point [True]
374 Text feature [131] present in test data point [True]
407 Text feature [1016] present in test data point [True]
439 Text feature [006] present in test data point [True]
441 Text feature [015] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [201]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.1289 0.5317 0.0024 0.1333 0.0192 0.0
816 0.0997 0.0018 0.0013]]
Actual Class : 1
--------------------------------------------------
12 Text feature [132] present in test data point [True]
19 Text feature [113] present in test data point [True]
433 Text feature [100] present in test data point [True]
474 Text feature [10] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (Tfidf)

In [202]:

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

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

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.263405769908992
for n_estimators = 100 and max depth =  10
Log Loss : 1.2578901687141508
for n_estimators = 200 and max depth =  5
Log Loss : 1.2591669904162979
for n_estimators = 200 and max depth =  10
Log Loss : 1.250677039901095
for n_estimators = 500 and max depth =  5
Log Loss : 1.241502698010277
for n_estimators = 500 and max depth =  10
Log Loss : 1.2432372130531888
for n_estimators = 1000 and max depth =  5
Log Loss : 1.237643902477869
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2398529881379918
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2340423917746326
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2361678615667273
For values of best estimator =  2000 The train log loss is: 0.835553332
0874516
For values of best estimator =  2000 The cross validation log loss is:
1.2340423917746326
For values of best estimator =  2000 The test log loss is: 1.1307085302
27666
```

4.5.2. Testing model with best hyper parameters (Tfidf)

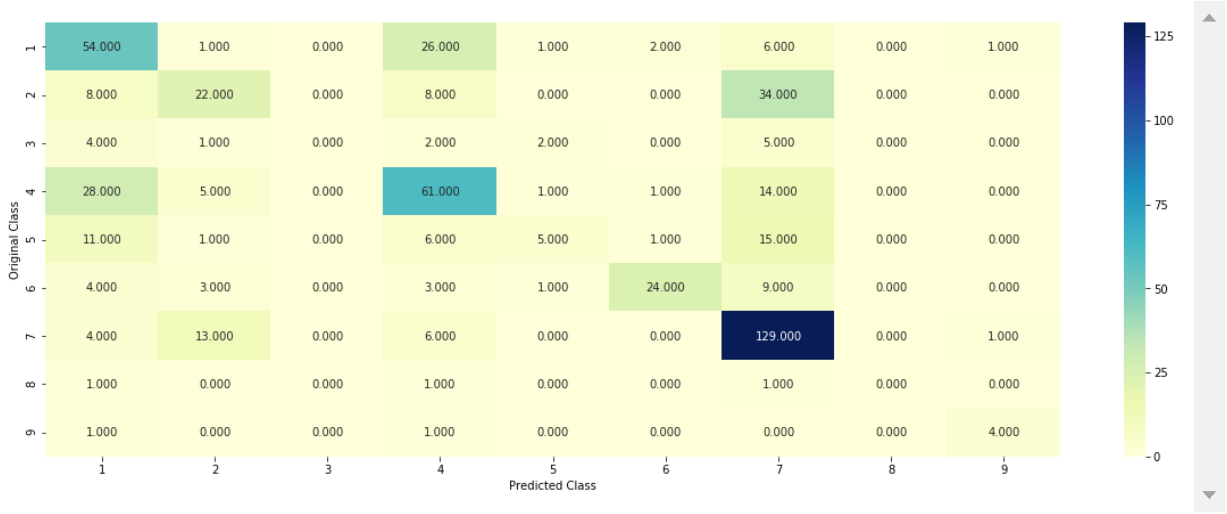
In [203]:

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

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
```

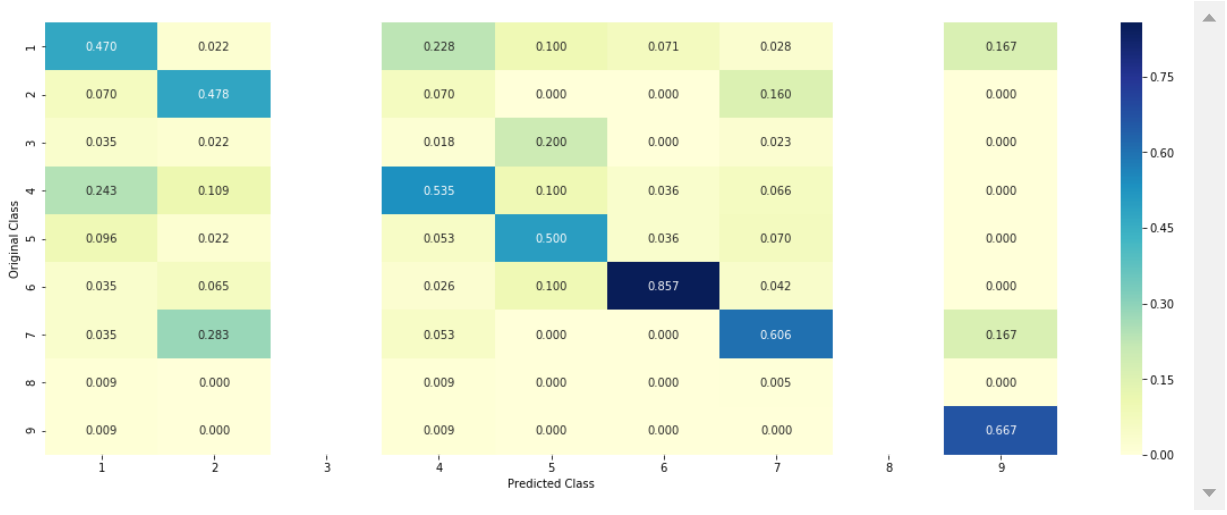
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
 n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_
y, clf)
```

```
Log loss : 1.2340423917746324
Number of mis-classified points : 0.43796992481203006
-------------------- Confusion matrix --------------------
```

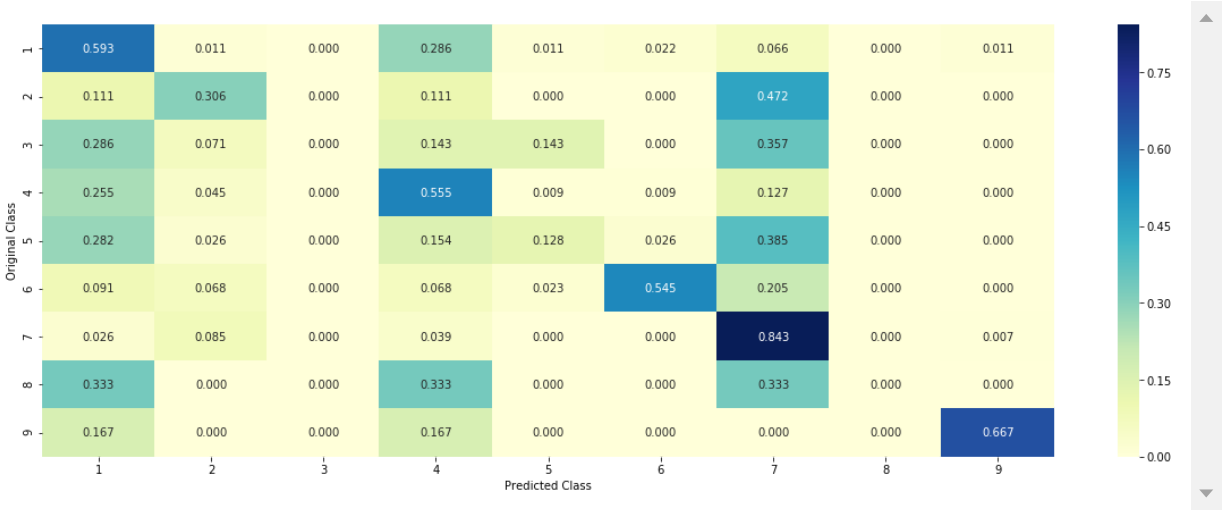



```
-------------------- Precision matrix (Columm Sum=1) ------------------
--
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [204]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.009  0.3744 0.0082 0.0068 0.0221 0.0
164 0.5606 0.0019 0.0007]]
Actual Class : 7
--------------------------------------------------
12 Text feature [103] present in test data point [True]
24 Text feature [1000] present in test data point [True]
43 Text feature [035] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [207]:
```
test_point_index = 55
no_feature = 55
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3262 0.066  0.0182 0.3663 0.0633 0.0
595 0.07   0.013  0.0176]]
Actuall Class : 1
--------------------------------------------------
Out of the top  55  features  0 are present in query point
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.5.3. Hyper paramter tuning (With Response Coding)

In [208]:
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=
clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.304501901748124
for n_estimators = 10 and max depth =  3
Log Loss : 1.8347426001862053
for n_estimators = 10 and max depth =  5
Log Loss : 1.5123074711611129
for n_estimators = 10 and max depth =  10
Log Loss : 1.9959525859802574
for n_estimators = 50 and max depth =  2
Log Loss : 1.8302792376410695
for n_estimators = 50 and max depth =  3
Log Loss : 1.5072479124512732
for n_estimators = 50 and max depth =  5
Log Loss : 1.4276756042162067
for n_estimators = 50 and max depth =  10
Log Loss : 1.7700865984508753
for n_estimators = 100 and max depth =  2
Log Loss : 1.66897503223448
for n_estimators = 100 and max depth =  3
Log Loss : 1.5820082920388532
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.394487906028632
for n_estimators = 100 and max depth =  10
Log Loss : 1.7177157838157455
for n_estimators = 200 and max depth =  2
Log Loss : 1.7524028951590975
for n_estimators = 200 and max depth =  3
Log Loss : 1.6200192752394231
for n_estimators = 200 and max depth =  5
Log Loss : 1.464164070511066
for n_estimators = 200 and max depth =  10
Log Loss : 1.6980196193473056
for n_estimators = 500 and max depth =  2
Log Loss : 1.8067777344637066
for n_estimators = 500 and max depth =  3
Log Loss : 1.6708807236504288
for n_estimators = 500 and max depth =  5
Log Loss : 1.4552357278559944
for n_estimators = 500 and max depth =  10
Log Loss : 1.7380342812691314
for n_estimators = 1000 and max depth =  2
Log Loss : 1.78617453328804
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6779813441996994
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4445498420188014
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7239776809481036
For values of best alpha =  100 The train log loss is: 0.05684492847830
641
For values of best alpha =  100 The cross validation log loss is: 1.394
4879060287083
For values of best alpha =  100 The test log loss is: 1.327662606666882
2
```

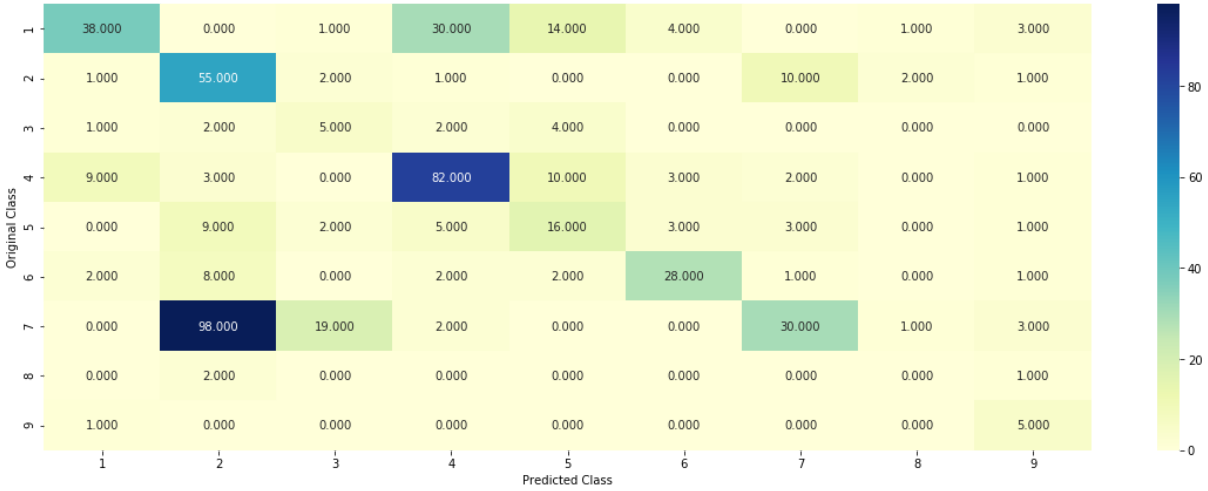
4.5.4. Testing model with best hyper parameters (Response Coding)

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [209]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.3944879060287083
Number of mis-classified points : 0.5131578947368421
-------------------- Confusion matrix --------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [210]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0099 0.6988 0.0805 0.0151 0.0161 0.0
326 0.1229 0.012  0.0121]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene 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
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Text is important feature
Gene is important feature

Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [211]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 9
Predicted Class Probabilities: [[0.0484 0.1627 0.1168 0.0452 0.0723 0.0
909 0.0568 0.1381 0.2688]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [212]:
```
from mlxtend.classifier import StackingClassifier
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(train_x_tfidf, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.62
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.035
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.512
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.141
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.274
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.654
```

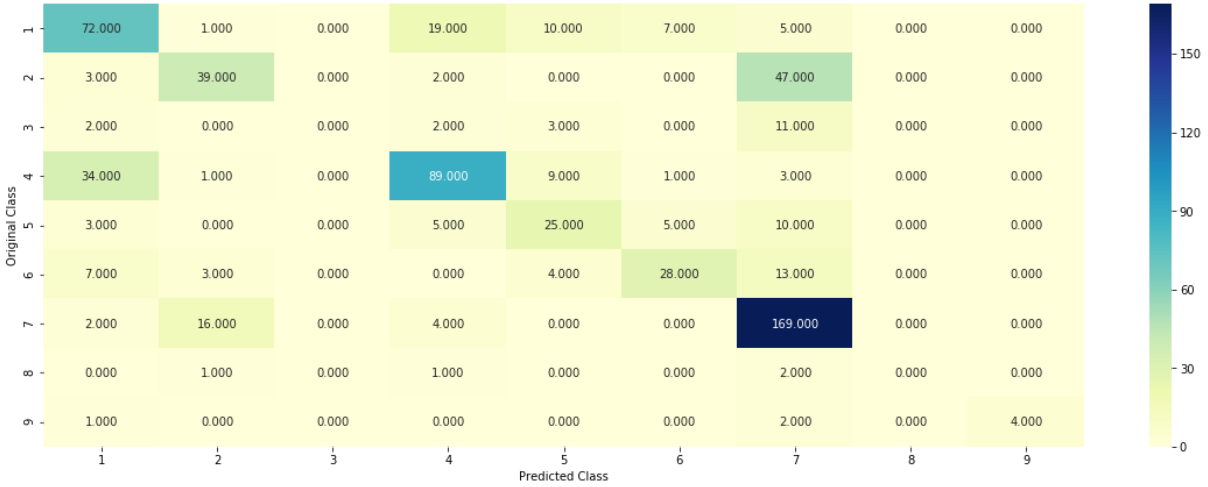
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

4.7.2 testing the model with the best hyper parameters

In [213]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_tfidf, train_y)

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

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

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

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

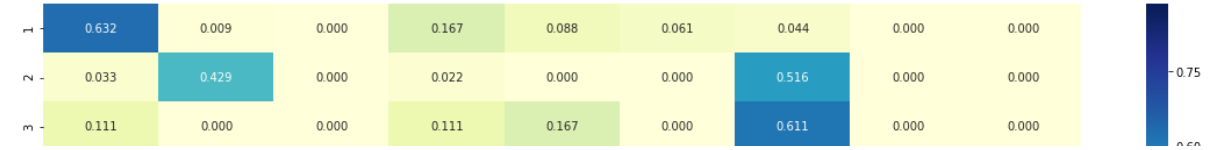
```
Log loss (train) on the stacking classifier : 0.5989068624567248
Log loss (CV) on the stacking classifier : 1.1414101234650664
Log loss (test) on the stacking classifier : 1.144557346907684
Number of missclassified point : 0.3593984962406015
-------------------- Confusion matrix --------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

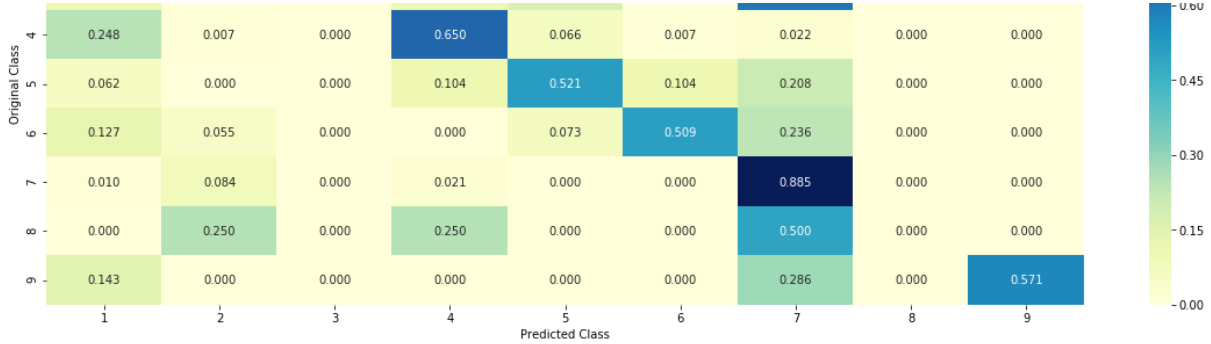


---------- Precision matrix (Columm Sum=1) ----------

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



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



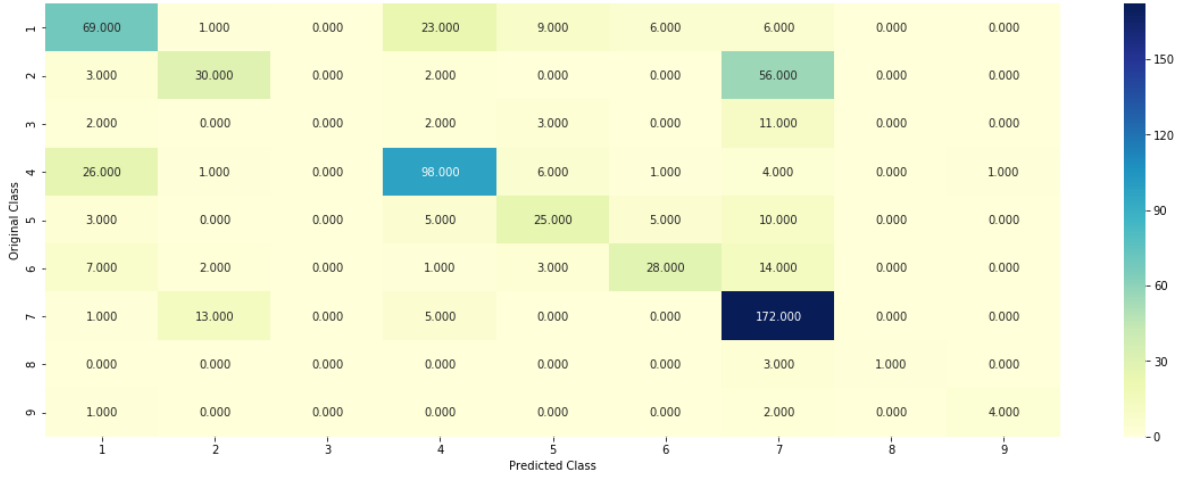
4.7.3 Maximum Voting classifier

In [214]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_tfidf, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_tfidf)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_tfidf)))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vcl
f.predict_proba(test_x_tfidf)))
print("Number of missclassified point :", np.count_nonzero((vclf.predic
t(test_x_tfidf)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfid
f))
```

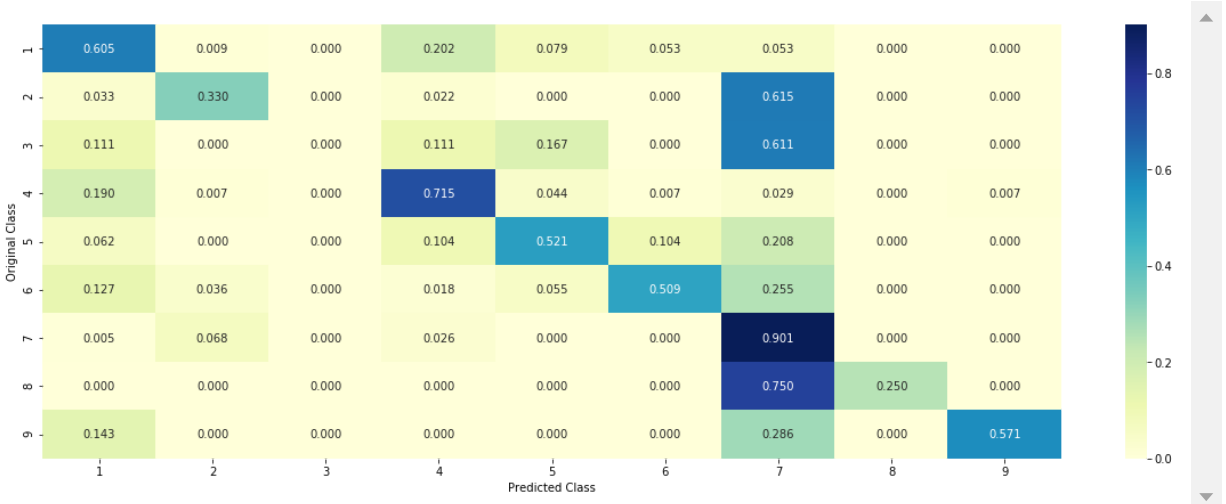
```
Log loss (train) on the VotingClassifier : 0.8479506934259647
Log loss (CV) on the VotingClassifier : 1.1760264898934252
Log loss (test) on the VotingClassifier : 1.1621521291283463
Number of missclassified point : 0.35789473684210527
--------------------- Confusion matrix --------------------
```




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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



Logistic Regression (count vectorizer class balancing)

In [112]:
```
train_df.head()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Out[112]:

	ID	Gene	Variation	Class	TEXT	gene_var	length
3113	3113	RAD51C	G264S	3	strong evidence overtly inactivating mutations...	RAD51C G264S	11517
1672	1672	FLT3	D835E	7	mutations receptor tyrosine kinases implicated...	FLT3 D835E	7746
2523	2523	BRCA1	L1854P	1	genetic screening breast ovarian cancer suscep...	BRCA1 L1854P	3538
298	298	CHEK2	E321K	4	maintenance genomic integrity depends coordina...	CHEK2 E321K	2649
2729	2729	BRAF	K601Q	7	noonan leopard cardiofaciocutaneous syndromes ...	BRAF K601Q	3351

In [114]:
```
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = CountVectorizer(ngram_range=(1,2),max_features=5000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [115]:
```
# don't forget to normalize every feature
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

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

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

In [116]:

```
#for gene variation feature
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
gene_var_vectorizer = CountVectorizer(ngram_range=(1,2),max_features=50
00)
train_gene_var_feature_onehotCoding = gene_var_vectorizer.fit_transform
(train_df['gene_var'])
# getting all the feature names (words)
train_gene_var_features= gene_var_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
gene_var_fea_dict = dict(zip(list(train_gene_var_features),train_gene_v
ar_fea_counts))


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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Total number of unique words in train data : 4365
```

In [117]:
```
# don't forget to normalize every feature
train_gene_var_feature_onehotCoding = normalize(train_gene_var_feature_
onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_gene_var_feature_onehotCoding = gene_var_vectorizer.transform(test
_df['gene_var'])
# don't forget to normalize every feature
test_gene_var_feature_onehotCoding = normalize(test_gene_var_feature_on
ehotCoding, axis=0)

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

In [118]:
```
train_x_onehotCoding = hstack((train_gene_var_feature_onehotCoding, tra
in_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_feature_onehotCoding, test_
text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_feature_onehotCoding, cv_text_f
eature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [119]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
```

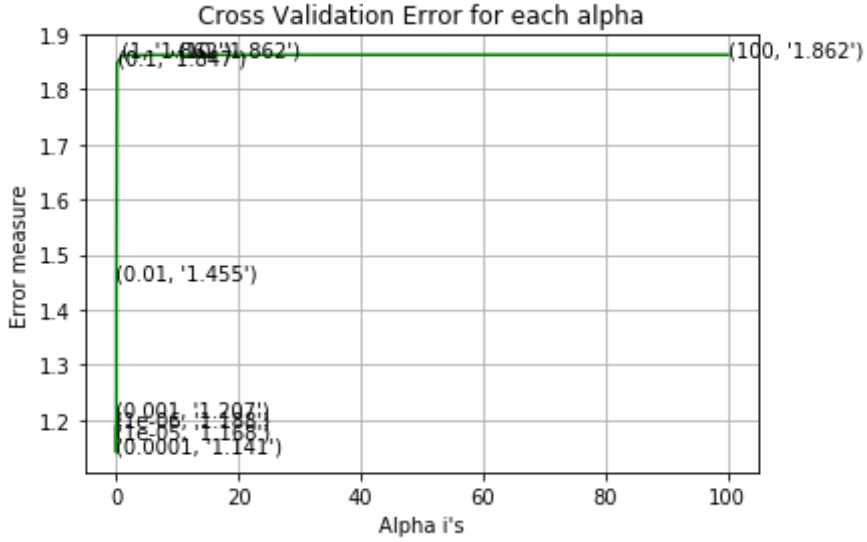
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Log Loss : 1.1879207941285095
for alpha = 1e-05
Log Loss : 1.1680638376912686
for alpha = 0.0001
Log Loss : 1.141212703006818
for alpha = 0.001
Log Loss : 1.2071105368846502
for alpha = 0.01
Log Loss : 1.4546716044688452
for alpha = 0.1
Log Loss : 1.8469972784627402
for alpha = 1
Log Loss : 1.8615270858844613
for alpha = 10
Log Loss : 1.8621181001567928
for alpha = 100
Log Loss : 1.862174353872471
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4561139784635551
For values of best alpha =  0.0001 The cross validation log loss is: 1.141212703006818
For values of best alpha =  0.0001 The test log loss is: 1.127841433035
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0819
```

4.3.1.2. Testing the model with best hyper paramters

In [120]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.141212703006818
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Logistic Regression(countvectorizer, unigram and bigram, without class balancing)

In [121]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

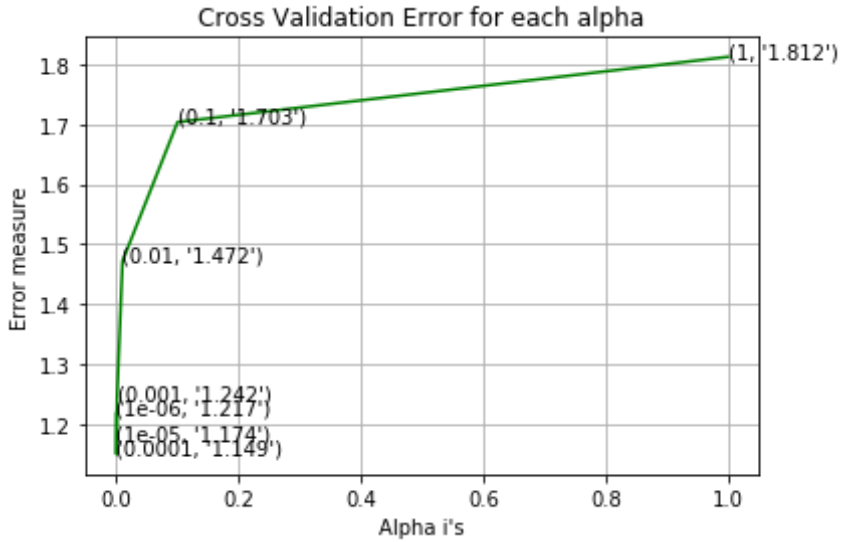
```
#------------------------------



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

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2174350063506203
for alpha = 1e-05
Log Loss : 1.1742181496855133
for alpha = 0.0001
Log Loss : 1.1491456443919505
for alpha = 0.001
Log Loss : 1.2420914811025277
for alpha = 0.01
Log Loss : 1.4715567181624394
for alpha = 0.1
Log Loss : 1.7034754016005542
for alpha = 1
Log Loss : 1.8123582207337083
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
For values of best alpha =  0.0001 The train log loss is: 0.45700081689
722694
For values of best alpha =  0.0001 The cross validation log loss is: 1.
1491456443919505
For values of best alpha =  0.0001 The test log loss is: 1.130018786441
2983
```

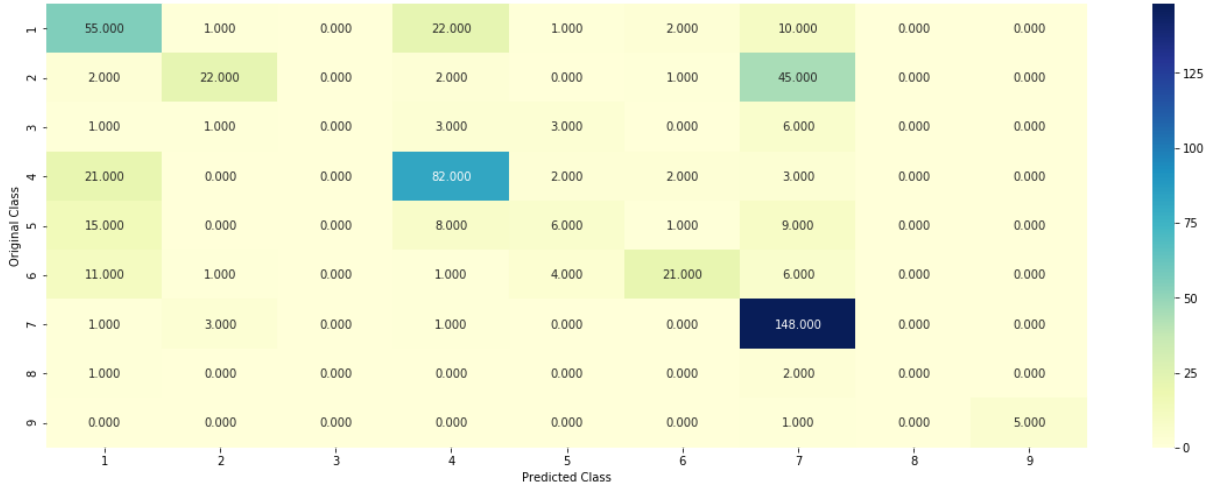
4.3.2.2. Testing model with best hyper parameters

In [122]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

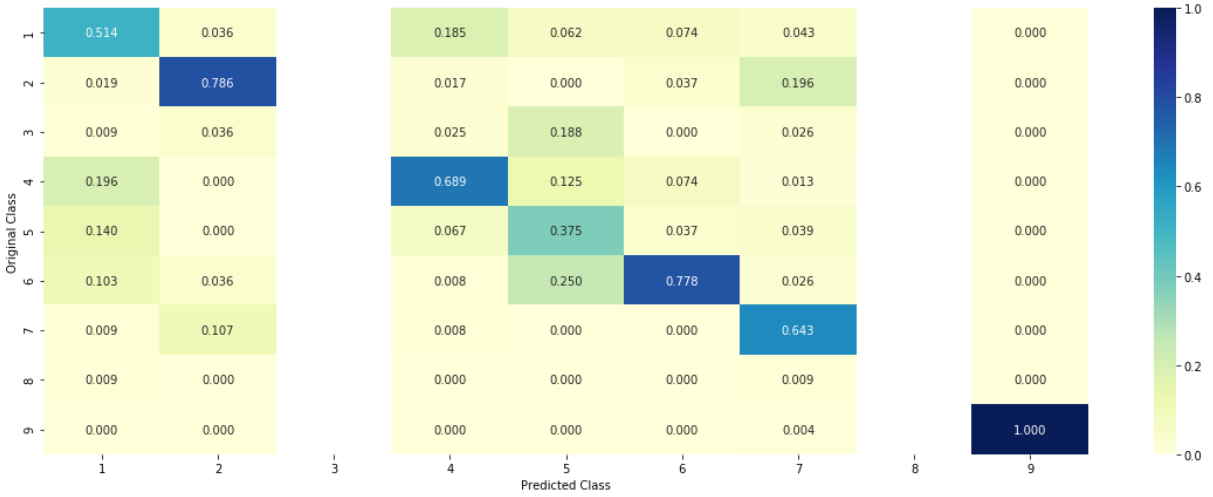
```
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.1491456443919505
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```

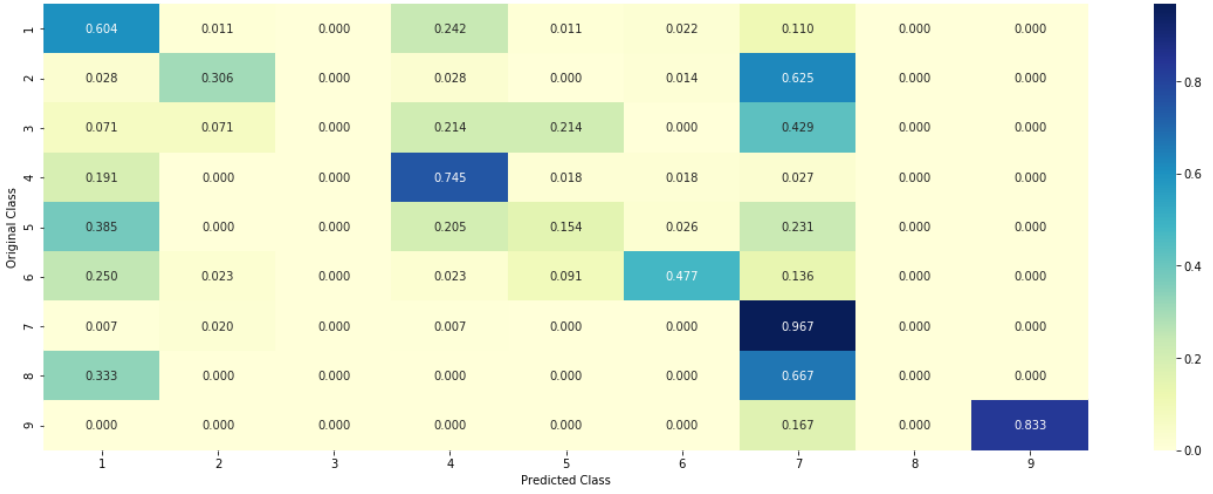



```
-------------------- Precision matrix (Columm Sum=1) ------------------
--
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Observations and Procedures

In [124]:
```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=['Model','Vectorizer','Train loss','Test loss','Cv loss',
'% misclassifed points','Stability']
x.add_row(['Naive Bayes','Tfidf','0.70','1.205','1.253','36.65%','Stabl
e'])
x.add_row(['KNN','Tfidf','0.63','1.033','1.037','36.27','Stable'])
x.add_row(['Logistic regression(class balancing)','Tfidf','0.73','1.05
3','1.024','37.96','Stable'])
x.add_row(['Logistic regression(without class balancing)','Tfidf','0.45
3','1.088','1.033','36.27','Unstable'])
x.add_row(['SVM','Tfidf','0.663','1.134','1.100','38.15','Stable'])
x.add_row(['RandomForest','Tfidf','0.835','1.234','1.307','43.79','Stab
le'])
x.add_row(['RandomForest','responeCoding','0.056','1.39','1.377','51.3
1','Unstable'])
x.add_row(['Stacking Model','TFidf','0.598','1.414','1.144','35.93','Un
stable'])
x.add_row(['Voting Classifier','Tfidf','0.847','1.176','1.162','35.78',
'Stable'])
x.add_row(['LogisticRegression(ClassBalancing)','CountVectorizer','0.45
6','1.141','1.127','35.90','Unstable'])
x.add_row(['LogisticRegression(withoutClassBalancing)','CountVectorize
r','0.457','1.149','1.130','36.278','Unstable'])
print(x)
```

```
+-----------------------------------------------+-----------------+-----
-------+-----------+---------+----------------------+-----------+
|                     Model                     |    Vectorizer   | Trai
n loss | Test loss | Cv loss | % misclassifed points | Stability |
+-----------------------------------------------+-----------------+-----
-------+-----------+---------+----------------------+-----------+
|                  Naive Bayes                  |      Tfidf      |
0.70    |   1.205   |  1.253  |        36.65%        |   Stable  |
|                    KNN                        |      Tfidf      |
0.63    |   1.033   |  1.037  |        36.27         |   Stable  |
|     Logistic regression(class balancing)      |      Tfidf      |
0.73    |   1.053   |  1.024  |        37.96         |   Stable  |
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0.73    |   1.053   |  1.024  |         37.96         |   Stable  |
| Logistic regression(without class balancing) |      Tfidf      |   0.
453    |   1.088   |  1.033  |         36.27         |  Unstable |
|                     SVM                      |      Tfidf      |   0.
663    |   1.134   |  1.100  |         38.15         |   Stable  |
|                  RandomForest                |      Tfidf      |   0.
835    |   1.234   |  1.307  |         43.79         |   Stable  |
|                  RandomForest                |  responeCoding  |   0.
056    |    1.39   |  1.377  |         51.31         |  Unstable |
|                Stacking Model                |      TFidf      |   0.
598    |   1.414   |  1.144  |         35.93         |  Unstable |
|              Voting Classifier               |      Tfidf      |   0.
847    |   1.176   |  1.162  |         35.78         |   Stable  |
|      LogisticRegression(ClassBalancing)      | CountVectorizer |   0.
456    |   1.141   |  1.127  |         35.90         |  Unstable |
|  LogisticRegression(withoutClassBalancing)   | CountVectorizer |   0.
457    |   1.149   |  1.130  |         36.278        |  Unstable |
+----------------------------------------------+-----------------+-----
-------+-----------+---------+-----------------------+-----------+
```

We have applied some feature engineering like merging gene and variation and length of text as a new feature but it's difficult to decrease the test and cv logloss to less than 1.

In our logistic Regression(without class balancing) model the train log loss is great but the difference between train log loss and test/cv log loss is high here out model may overfits.

In terms of train log loss our logistic regression(without class balancing) model is good but overall logistic regression(with class balancing) is out best model because test/cv logloss are comparatively low as compared to others and the difference between train and test/cv logloss is also low.

In [ ]:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD