Personalized_Cancer_Diagnosis_Case_Study

February 9, 2019

Personalized cancer diagnosis

1. Business Problem

1.1. Description
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)
Download training_variants.zip and training_text.zip from Kaggle.
Context:
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462
Problem statement :
Classify the given genetic variations/mutations based on evidence from text-based clinical literature.
1.2. Source/Useful Links
Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.

- Both these data files are have a common column called ID
- Data file's information:

```
<li>
training_variants (ID , Gene, Variations, Class)
</li>
<li>
training_text (ID, Text)
</li>
```

2.1.2. Example Data Point
training_variants
ID,Gene,Variation,Class 0,FAM58A,Truncating Mutations,1 1,CBL,W802*,2 2,CBL,Q249E,2 ...
training_text
ID,Text 0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem
2.2.1. Type of Machine Learning Problem

```
There are nine different classes a genetic mutation can be classified into => Multi cla
```

2.2.2. Performance Metric
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation
Metric(s): * Multi class log-loss * Confusion matrix
2.2.3. Machine Learing Objectives and Constraints
Objective: Predict the probability of each data-point belonging to each of the nine classes.
Constraints:

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

2.3. Train, CV and Test Datasets
Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import re
        import time
        import warnings
        import numpy as np
        import seaborn as sns
        from collections import Counter, defaultdict
        from nltk.corpus import stopwords
        from sklearn.decomposition import TruncatedSVD
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
        from sklearn.manifold import TSNE
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import confusion_matrix, normalized_mutual_info_score
        from sklearn.metrics.classification import accuracy_score, log_loss
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.linear_model import SGDClassifier
        from imblearn.over_sampling import SMOTE
        from collections import Counter
        from scipy.sparse import hstack
        from sklearn.multiclass import OneVsRestClassifier
        from sklearn.svm import SVC

        from sklearn.calibration import CalibratedClassifierCV
        from sklearn.naive_bayes import MultinomialNB, GaussianNB
        from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold

        import math
        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 [2]: data_variants = pd.read_csv('G:/Applied AI case study/personalized_cancer_diagnosis/tra
        print('Number of data points : ', data_variants.shape[0])
        print('Number of features : ', data_variants.shape[1])
        print('Features : ', data_variants.columns.values)
        data_variants.head()
```

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

```
Out[2]:    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
Fields are
<ul>
    <li><b>ID : </b>the id of the row used to link the mutation to the clinical evidence</li>
    <li><b>Gene : </b>the gene where this genetic mutation is located </li>
    <li><b>Variation : </b>the aminoacid change for this mutations </li>
    <li><b>Class :</b> 1-9 the class this genetic mutation has been classified on</li>
</ul>
```

3.1.2. Reading Text Data

```
In [3]: # note the seprator in this file
        data_text =pd.read_csv("G:/Applied AI case study/personalized_cancer_diagnosis/trainin
        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[3]:

	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 [4]:
```
# loading stop words from nltk library
from nltk.stem import SnowballStemmer
import nltk
stop_words = set(stopwords.words('english'))
sno = nltk.stem.SnowballStemmer('english')


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

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

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

In [5]:
```
# Text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
Time took for preprocessing the text : 772.4802785365796 seconds
```

In [6]:
```
# Merging both gene_variations and text data based on ID
result = pd.merge(data_variants, data_text,on='ID', how='left')
result.head()
```

```
Out[6]:    ID    Gene             Variation  Class  \
        0   0  FAM58A  Truncating Mutations      1
        1   1     CBL                 W802*      2
        2   2     CBL                 Q249E      2
        3   3     CBL                 N454D      3
        4   4     CBL                 L399V      4

                                                  TEXT
        0  cyclin depend kinas cdks regul varieti fundame...
        1  abstract background non small cell lung cancer...
        2  abstract background non small cell lung cancer...
        3  recent evid demonstr acquir uniparent disomi a...
        4  oncogen mutat monomer casita b lineag lymphoma...
```

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 [7]: result.Gene = result.Gene.str.replace('\s+', '_')
        result.Variation = result.Variation.str.replace('\s+', '_')
        y_true = result[['Class']]
        x_true = result.drop(['Class'], axis=1)

In [8]: # Split the data into test and train by maintaining same distribution of output varaib
        x_train, x_test, y_train, y_test = train_test_split(x_true, y_true, stratify=y_true, te

        # Split the train data into train and cross validation by maintaining same distributio
        x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train, tes
```

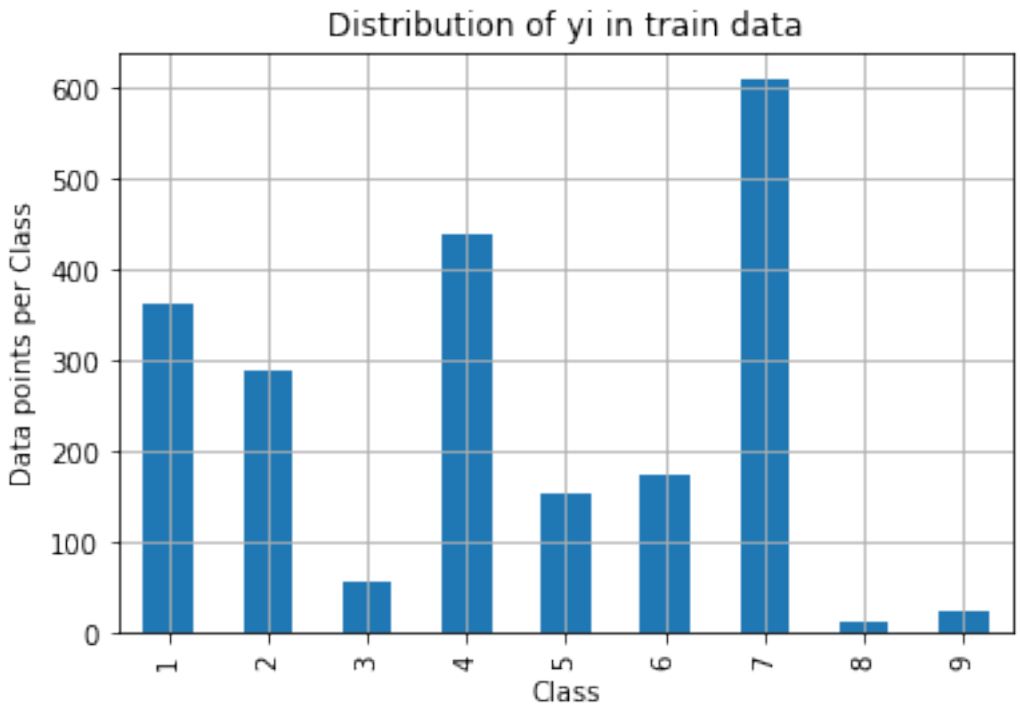
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

```
In [11]: print('Number of data points in train data:', x_train.shape[0])
         print('Number of data points in test data:', x_test.shape[0])
         print('Number of data points in cross validation data:', x_cv.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
```

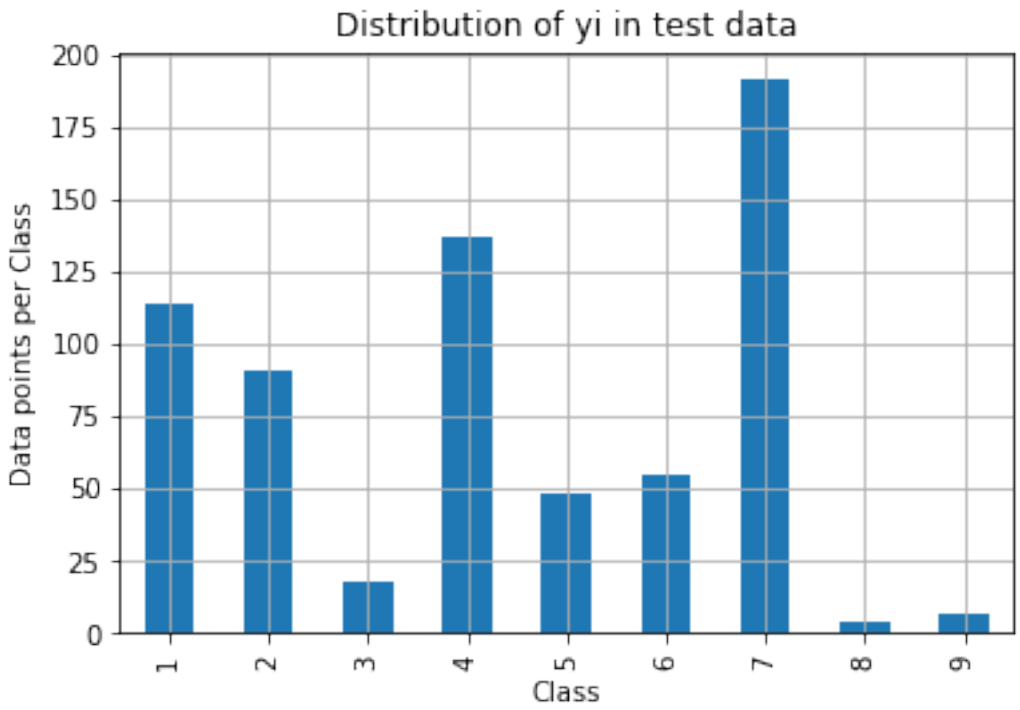
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [13]: # it returns a dict, keys as class labels and values as the number of data points in
         train_class_distribution = y_train['Class'].value_counts().sortlevel() # sortlevel()
         test_class_distribution = y_test['Class'].value_counts().sortlevel()
         cv_class_distribution = y_cv['Class'].value_counts().sortlevel()

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

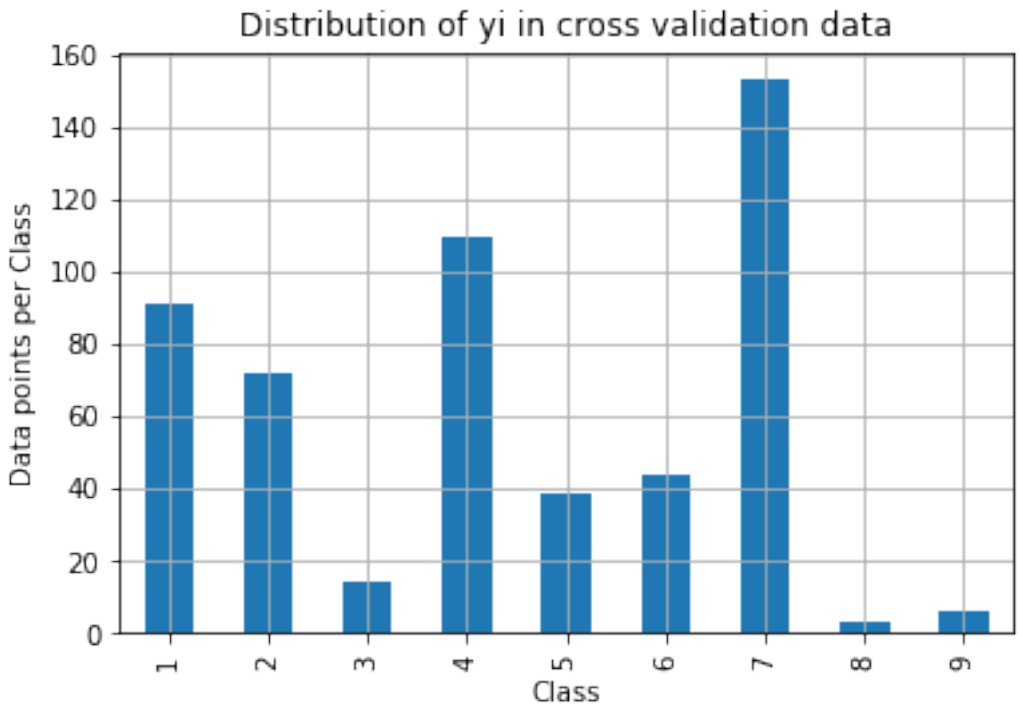
```
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
# -(train_class_distribution.values): the minus sign will give us in decreasing 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_distribution.values[i]

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

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




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




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




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

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the '9' class probabilites randomly such that they sum to 1.

```
In [16]: def plot_matrix(matrix,labels):
             plt.figure(figsize=(20,7))
             sns.heatmap(matrix, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.show()

         # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
```

```
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

    recall_table =(((C.T)/(C.sum(axis=1))).T)
    # How did we calculateed recall_table :
    # divide each element of the confusion matrix with the sum of elements in that co
    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]
    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    precision_table =(C/C.sum(axis=0))
    # How did we calculateed precision_table :
    # divide each element of the confusion matrix with the sum of elements in that ro
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    print()
    print("-"*20, "Confusion matrix", "-"*20)
    plot_matrix(C,labels)

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plot_matrix(precision_table,labels)

    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plot_matrix(recall_table,labels)
```

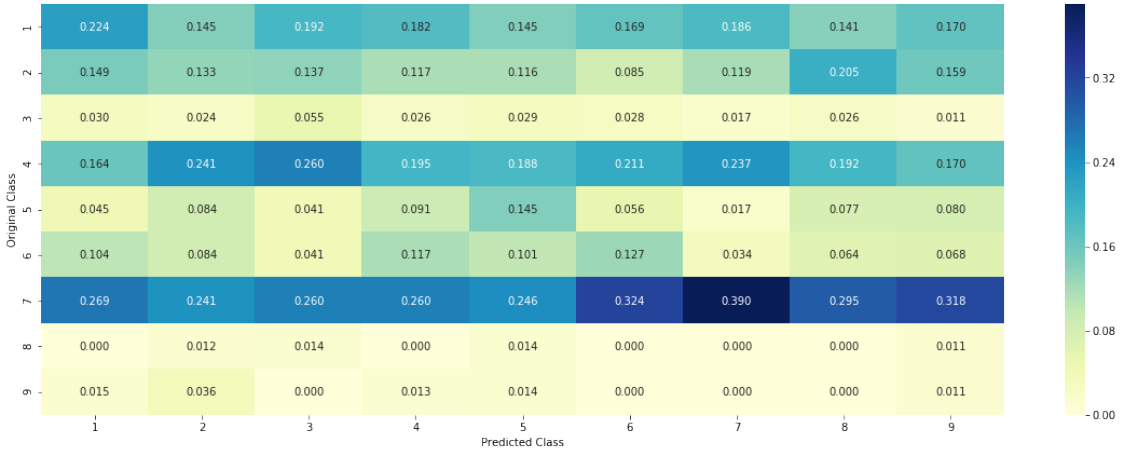
In [17]:
```
# We need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = x_test.shape[0]
cv_data_len = x_cv.shape[0]

# 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_predicte


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

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

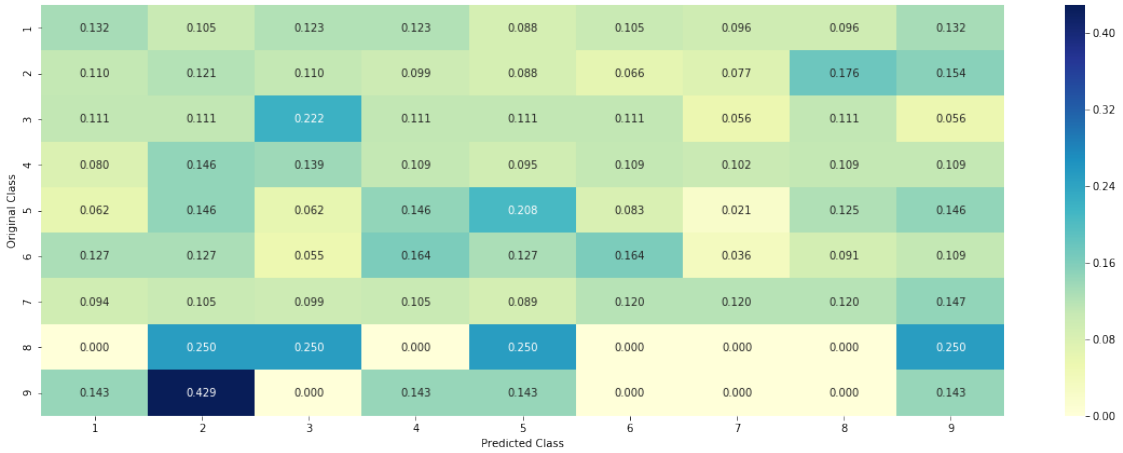
```
Log loss on Cross Validation Data using Random Model 2.48242936395
Log loss on Test Data using Random Model 2.49511372872
```

```
-------------------- Confusion matrix --------------------
```


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




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



3.3 Univariate Analysis

```
In [29]: # code for response coding with Laplace smoothing.
         # alpha : used for laplace smoothing
         # feature: ['gene', 'variation']
         # df: ['x_train', 'x_test', 'x_cv']
         # algorithm
         # ----------
         # Consider all unique values and the number of occurances of given feature in train d
         # build a vector (1*9) , the first element = (number of times it occured in class1 +
         # gv_dict is like a look up table, for every gene it store a (1*9) representation of
         # for a value of feature in df:
         # if it is in train data:
```

```
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

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

                # cls_cnt.shape[0](numerator) will contain the number of time that partic
                vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

    # Get Gene variation feature
    def get_gv_feature(alpha, feature, df):
        # print(gv_dict)
        #     {'BRCA1': [0.20075757575757575, 0.037878787878788, 0.068181818181818177,
        #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
        #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181
        #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
        #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
        #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0
        #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
        #      ...
        #     }
        gv_dict = get_gv_fea_dict(alpha, feature, df)
        # value_count is similar in get_gv_fea_dict
        value_count = x_train[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

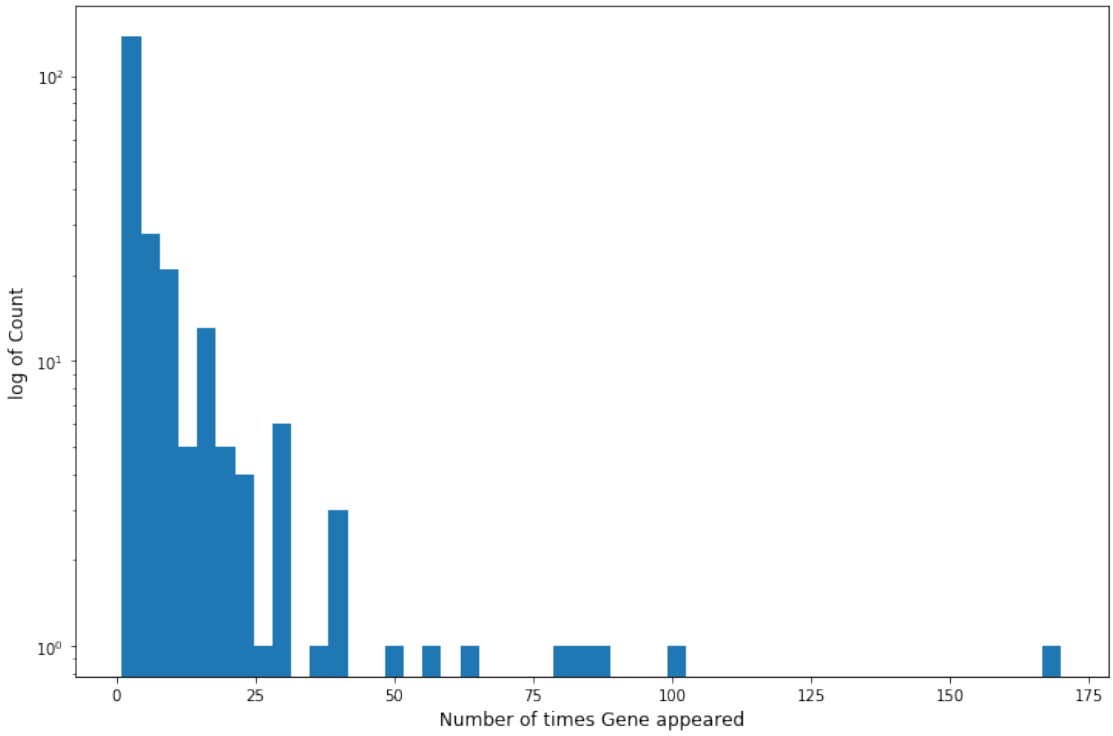
Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

```
In [9]: unique_genes = x_train['Gene'].value_counts()
        x_train['Gene'].value_counts().plot.bar()
        print('Number of Unique Genes :', unique_genes.shape[0])

Number of Unique Genes : 233
```

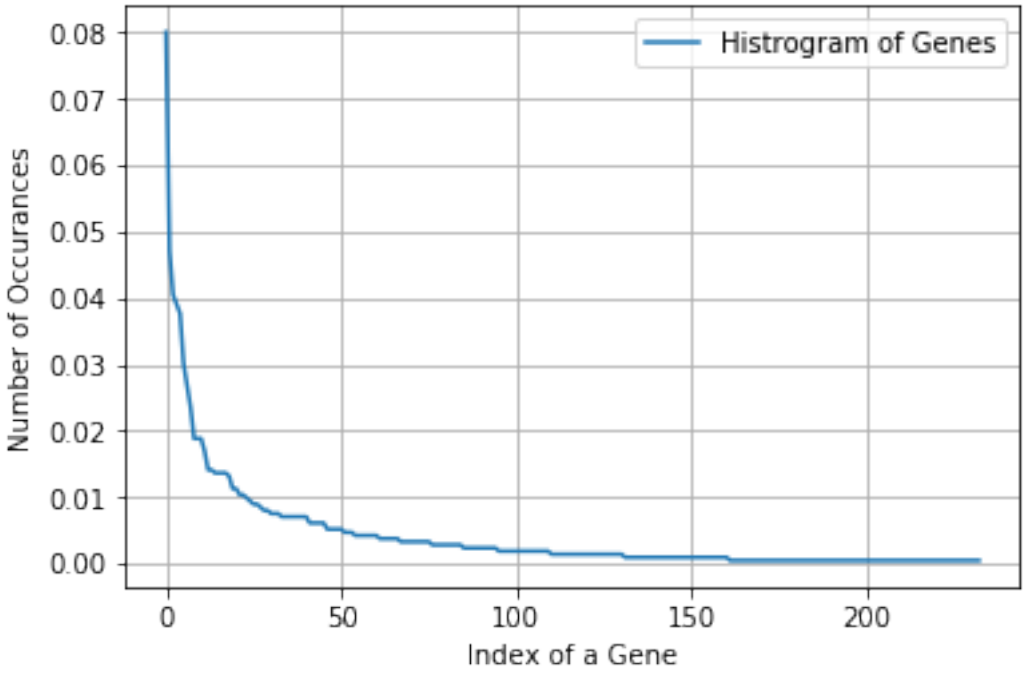
```
In [25]: train_genes = x_train.groupby('Gene')['Gene'].count()

         for i in [2, 5, 10, 20, 50, 100, 300]:
             print('Genes that appear less than {} times: {}%'.format(i, round((train_genes < i

         plt.figure(figsize=(12, 8))
         plt.hist(train_genes.values, bins=50, log=True)
         plt.xlabel('Number of times Gene appeared', fontsize=12)
         plt.ylabel('log of Count', fontsize=12)
         plt.show()

Genes that appear less than 2 times: 30.9%
Genes that appear less than 5 times: 59.23%
Genes that appear less than 10 times: 76.82%
Genes that appear less than 20 times: 89.27%
Genes that appear less than 50 times: 96.57%
Genes that appear less than 100 times: 99.14%
Genes that appear less than 300 times: 100.0%
```



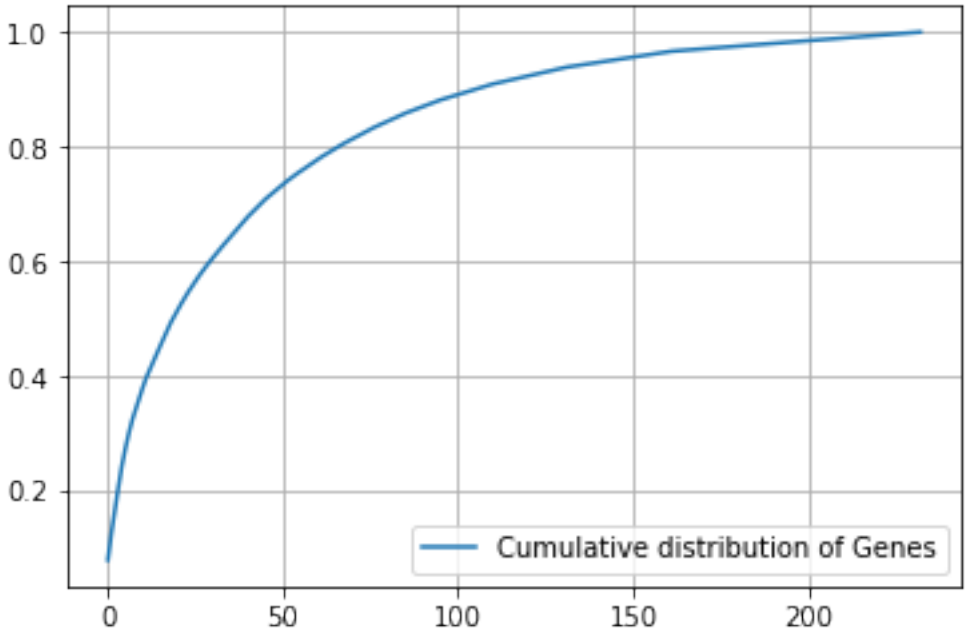

```
In [26]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the t
```

```
Ans: There are 233 different categories of genes in the train data, and they are distibuted as
```

```
In [27]: s = sum(unique_genes.values);
         h = unique_genes.values/s;
         plt.plot(h, label="Histrogram of Genes")
         plt.xlabel('Index of a Gene')
         plt.ylabel('Number of Occurances')
         plt.legend()
         plt.grid()
         plt.show()
```



In [28]:
```
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/

One hot Encoding

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

In [31]: print("train_gene_feature_responseCoding is converted feature using respone coding met

train_gene_feature_responseCoding is converted feature using respone coding method. The shape


In [32]: # one-hot encoding of Gene feature.
         gene_vectorizer = TfidfVectorizer()
         train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
         test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
         cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])

In [33]: train_gene_feature_onehotCoding

Out[33]: <2124x232 sparse matrix of type '<class 'numpy.float64'>'
                 with 2124 stored elements in Compressed Sparse Row format>

In [34]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding met
               train_gene_feature_onehotCoding.shape)

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

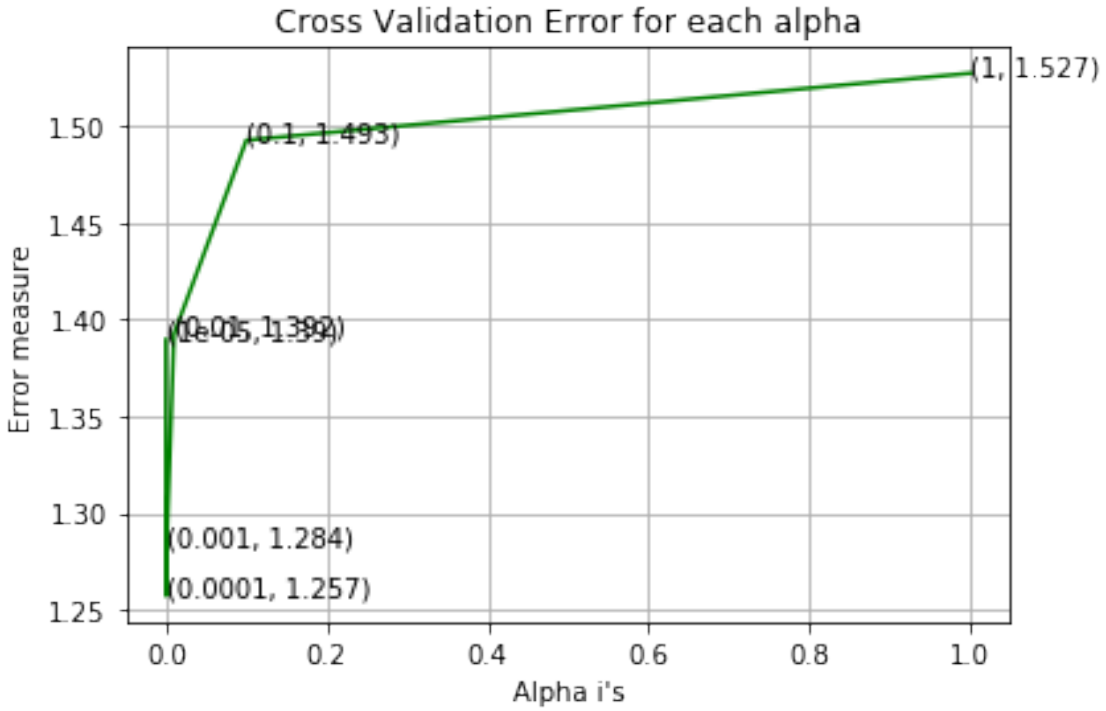
Q4. How good is this gene feature in predicting y_i?

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/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha],
```

```
            "The train log loss is:",
            log_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 validation log loss is:",
            log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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

```
For values of alpha =  1e-05 The log loss is: 1.3895142180772
For values of alpha =  0.0001 The log loss is: 1.2571578946778155
For values of alpha =  0.001 The log loss is: 1.283858562620601
For values of alpha =  0.01 The log loss is: 1.3923780764358225
For values of alpha =  0.1 The log loss is: 1.492530448447361
For values of alpha =  1 The log loss is: 1.5271261141759542
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0182913168115704
For values of best alpha =  0.0001 The cross validation log loss is: 1.2571578946778155
```

```
For values of best alpha =  0.0001 The test log loss is: 1.2078462444850886
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

```
In [22]: 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=x_test[x_test['Gene'].isin(list(set(x_train['Gene'])))].shape[0]
         cv_coverage=x_cv[x_cv['Gene'].isin(list(set(x_train['Gene'])))].shape[0]

         print('Ans\n1. In test data',test_coverage, 'out of',x_test.shape[0], ":",(test_covera
         print('2. In cross validation data',cv_coverage, 'out of ',x_cv.shape[0],":" ,(cv_cove
```

```
Q6. How many data points in Test and CV datasets are covered by the  237  genes in train datase
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 518 out of  532 : 97.36842105263158
```

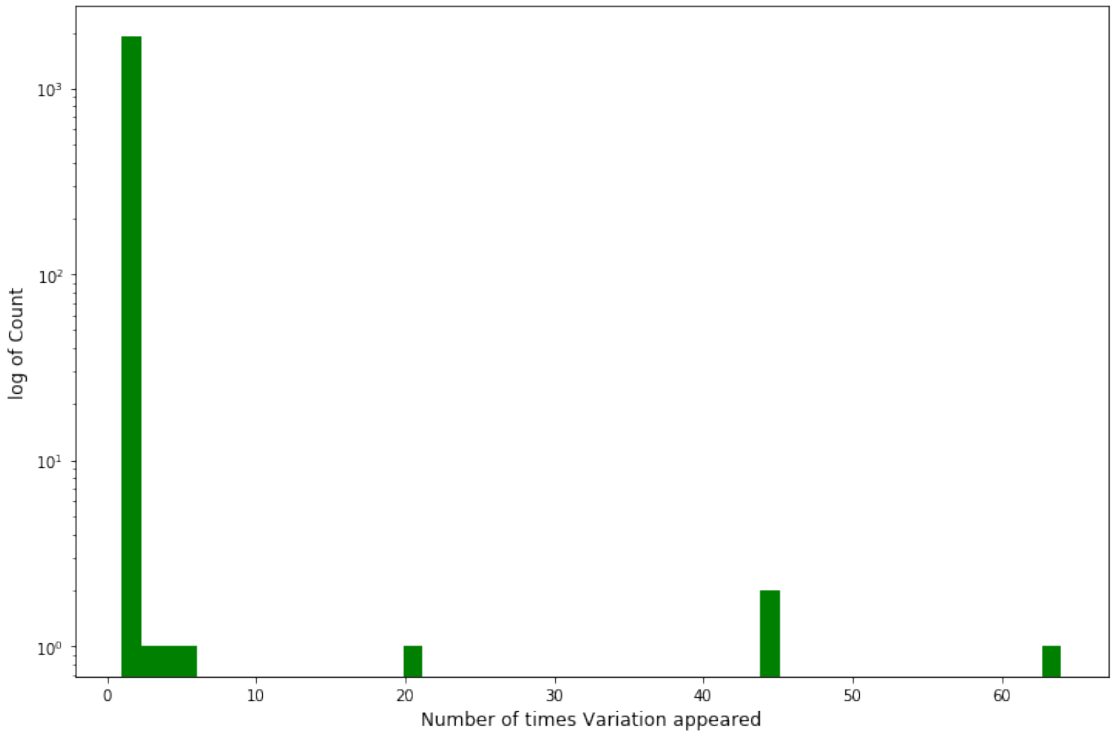
3.2.2 Univariate Analysis on Variation Feature
Q7. Variation, What type of feature is it ?
Ans. Variation is a categorical variable
Q8. How many categories are there?

```
In [10]: unique_variations = x_train['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
```

```
Number of Unique Variations : 1938
```

```
In [36]: train_variation = x_train.groupby('Variation')['Variation'].count()

         for i in [2,3,5,50,100]:
             print('Variation that appear less than {} times: {}%'.format(i, round((train_varia

         plt.figure(figsize=(12, 8))
         plt.hist(train_variation.values, bins=50, log=True, color='green')
         plt.xlabel('Number of times Variation appeared', fontsize=12)
         plt.ylabel('log of Count', fontsize=12)
         plt.show()
```

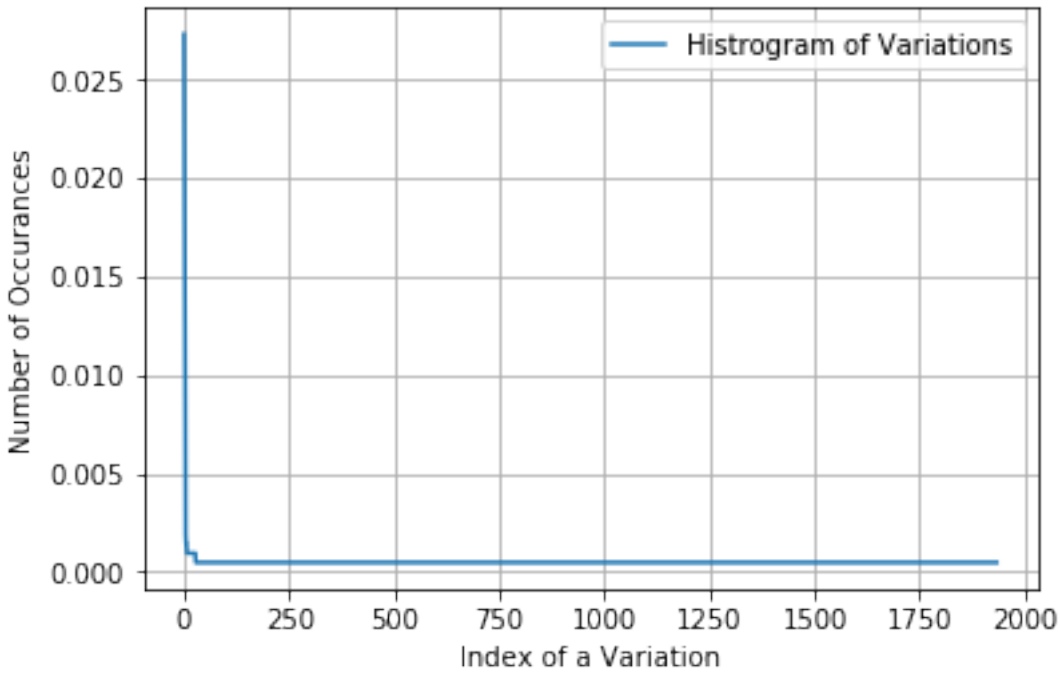
```
Variation that appear less than 2 times: 98.65%
Variation that appear less than 3 times: 99.64%
Variation that appear less than 5 times: 99.74%
Variation that appear less than 50 times: 99.95%
Variation that appear less than 100 times: 100.0%
```



- Majority of the times, the number of occurrences of the variations are less than 6.

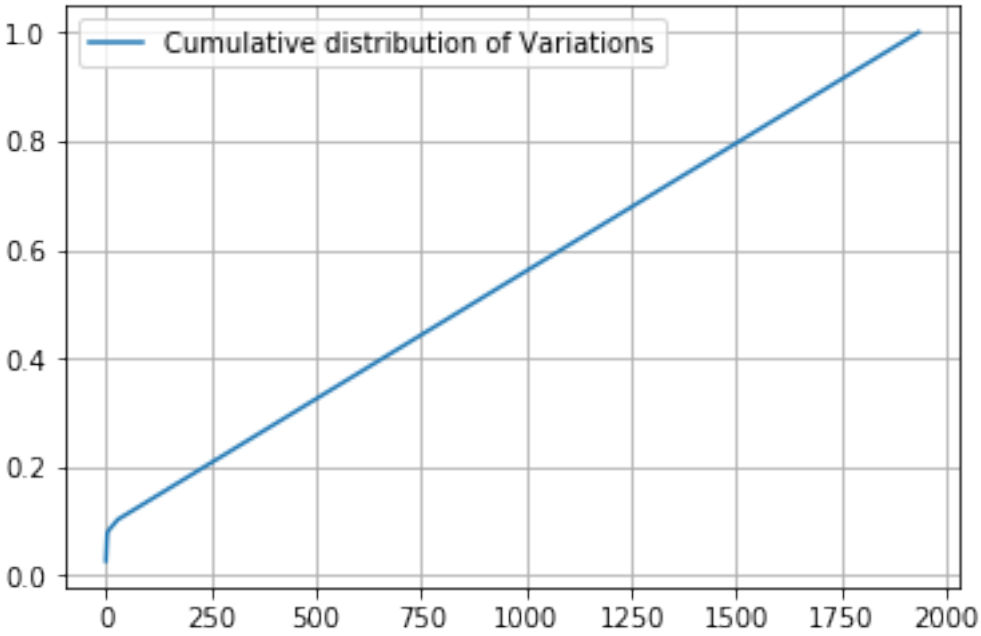
```
In [37]: print("Ans: There are", unique_variations.shape[0] ,
                "different categories of variations in the train data, and they are distibuted a

Ans: There are 1926 different categories of variations in the train data, and they are distibut
```

```
In [30]: s = sum(unique_variations.values);
         h = unique_variations.values/s;
         plt.plot(h, label="Histrogram of Variations")
         plt.xlabel('Index of a Variation')
         plt.ylabel('Number of Occurances')
         plt.legend()
         plt.grid()
         plt.show()
```




```
In [31]: c = np.cumsum(h)
         print(c)
         plt.plot(c,label='Cumulative distribution of Variations')
         plt.grid()
         plt.legend()
         plt.show()
```

```
[0.02730697 0.04943503 0.06920904 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

One hot Encoding

Response coding

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

```
In [38]: # alpha is used for laplace smoothing
         alpha = 1

         # train gene feature
         train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x

         # test gene feature
         test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_

         # cross validation gene feature
         cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_cv
```

```
In [39]: print("train_variation_feature_responseCoding is a converted feature using the respons
               train_variation_feature_responseCoding.shape)
```

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

```
In [40]: # one-hot encoding of variation feature.
         variation_vectorizer = TfidfVectorizer()
```

```
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['Va
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variatio
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'])
```

```
In [41]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot
         train_variation_feature_onehotCoding.shape)
```

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

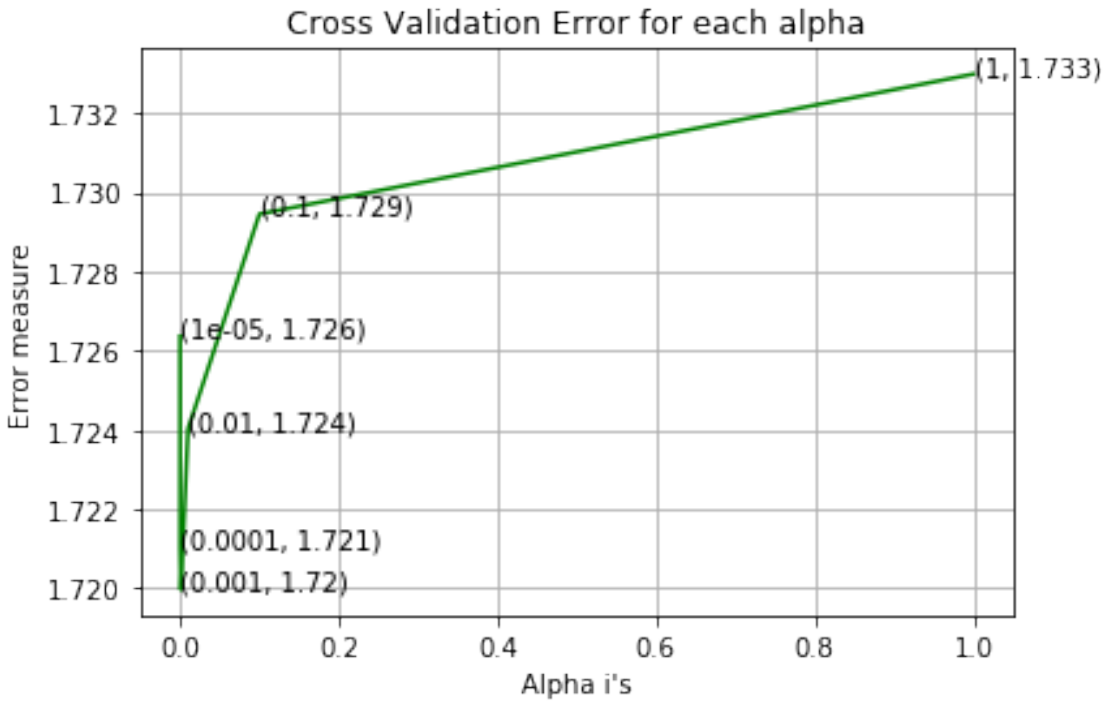
Q10. How good is this Variation feature in predicting y_i?
Let's build a model just like the earlier!

```
In [36]: alpha = [10 ** x for x in range(-5, 1)]
```

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

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

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


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

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

```
plt.show()


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

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

```
For values of alpha =  1e-05 The log loss is: 1.726366011044994
For values of alpha =  0.0001 The log loss is: 1.7210171101434846
For values of alpha =  0.001 The log loss is: 1.7199711703701446
For values of alpha =  0.01 The log loss is: 1.7240293115124958
For values of alpha =  0.1 The log loss is: 1.7294471826097995
For values of alpha =  1 The log loss is: 1.7329781232556747
```




```
For values of best alpha =  0.001 The train log loss is: 1.0451334878012042
For values of best alpha =  0.001 The cross validation log loss is: 1.7199711703701446
```

```
For values of best alpha =  0.001 The test log loss is: 1.7203386918678736
```

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 [37]: print("Q12. How many data points are covered by total ",
                 unique_variations.shape[0],
                 " genes in test and cross validation data sets?")
         test_coverage=x_test[x_test['Variation'].isin(list(set(x_train['Variation'])))].shape
         cv_coverage=x_cv[x_cv['Variation'].isin(list(set(x_train['Variation'])))].shape[0]
         print('Ans\n1. In test data',test_coverage, 'out of',x_test.shape[0], ":",(test_covera
         print('2. In cross validation data',cv_coverage, 'out of ',x_cv.shape[0],":" ,(cv_cove
```

```
Q12. How many data points are covered by total  1934  genes in test and cross validation data s
Ans
1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 60 out of  532 : 11.278195488721805
```

3.2.3 Univariate Analysis on Text Feature

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

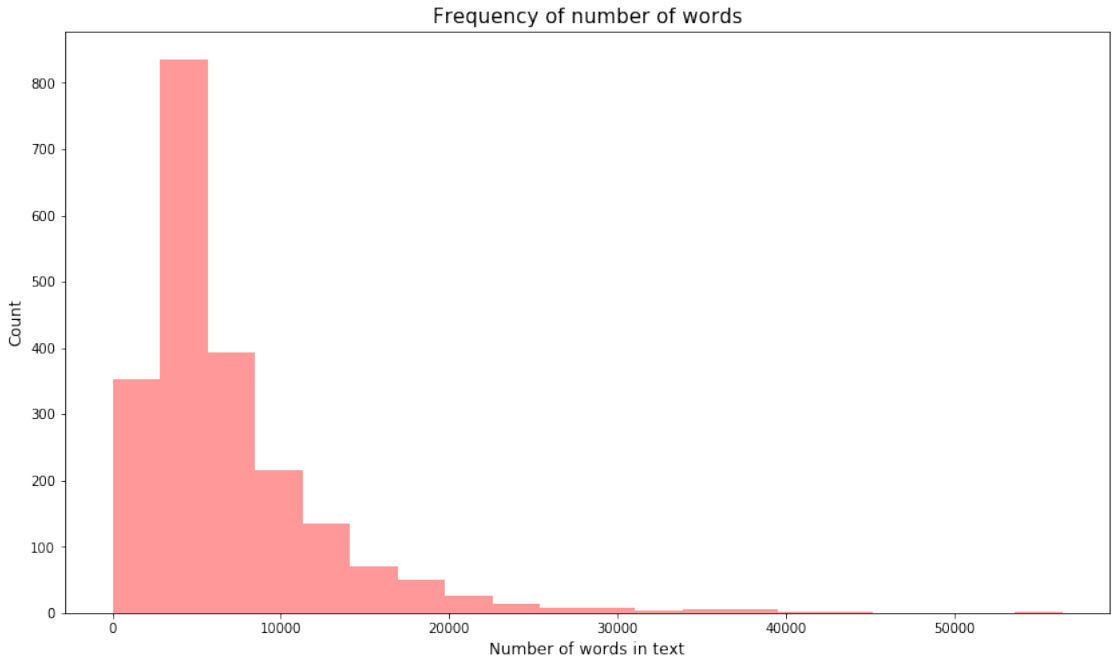
```
In [12]: x_train.head()
```

```
Out[12]:         ID    Gene Variation  \
         1797  1797      AR     L272F
         1217  1217  PIK3CA     I391M
         1495  1495   FGFR2     D101Y
         1645  1645    FLT3     F590G
         2346  2346    JAK2     R683G

                                                         TEXT
         1797  androgen receptor ar mutat associ androgen ins...
         1217  phosphatidylinositol 3 kinas pi3k pathway freq...
         1495  oncogen activ tyrosin kinas common mechan carc...
         1645  acut myeloid leukemia aml two cluster activ mu...
         2346  children syndrom great increas risk acut megak...
```

```
In [13]: x_train["Text_num_words"] = x_train["TEXT"].apply(lambda x: len(str(x).split()) )
         x_train["Text_num_chars"] = x_train["TEXT"].apply(lambda x: len(str(x)) )
```

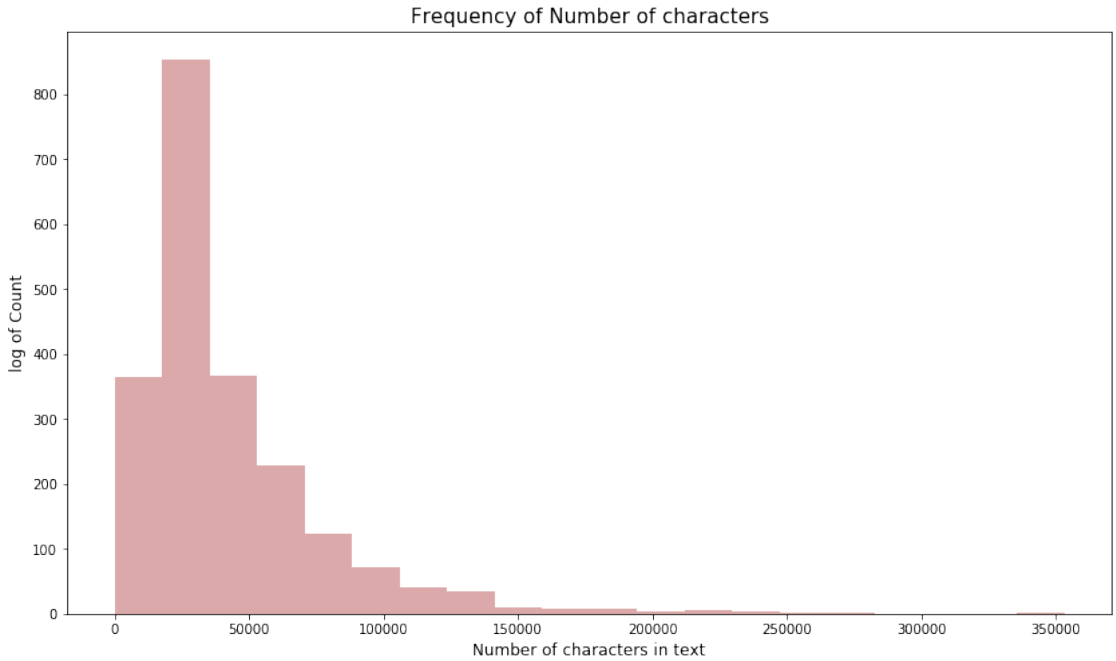
0.0.1 Let us look at the distribution of number of words in the text column.

```
In [15]: plt.figure(figsize=(14, 8))
         sns.distplot(x_train.Text_num_words.values, bins=20, kde=False, color='red')
         plt.xlabel('Number of words in text', fontsize=12)
         plt.ylabel('Count', fontsize=12)
         plt.title("Frequency of number of words", fontsize=15)
         plt.show()
```



- Majority of the word is around 4000 words. Now let us look at character level.

```
In [16]: plt.figure(figsize=(14, 8))
         sns.distplot(x_train.Text_num_chars.values, bins=20, kde=False, color='brown')
         plt.xlabel('Number of characters in text', fontsize=12)
         plt.ylabel('log of Count', fontsize=12)
         plt.title("Frequency of Number of characters", fontsize=15)
         plt.show()
```

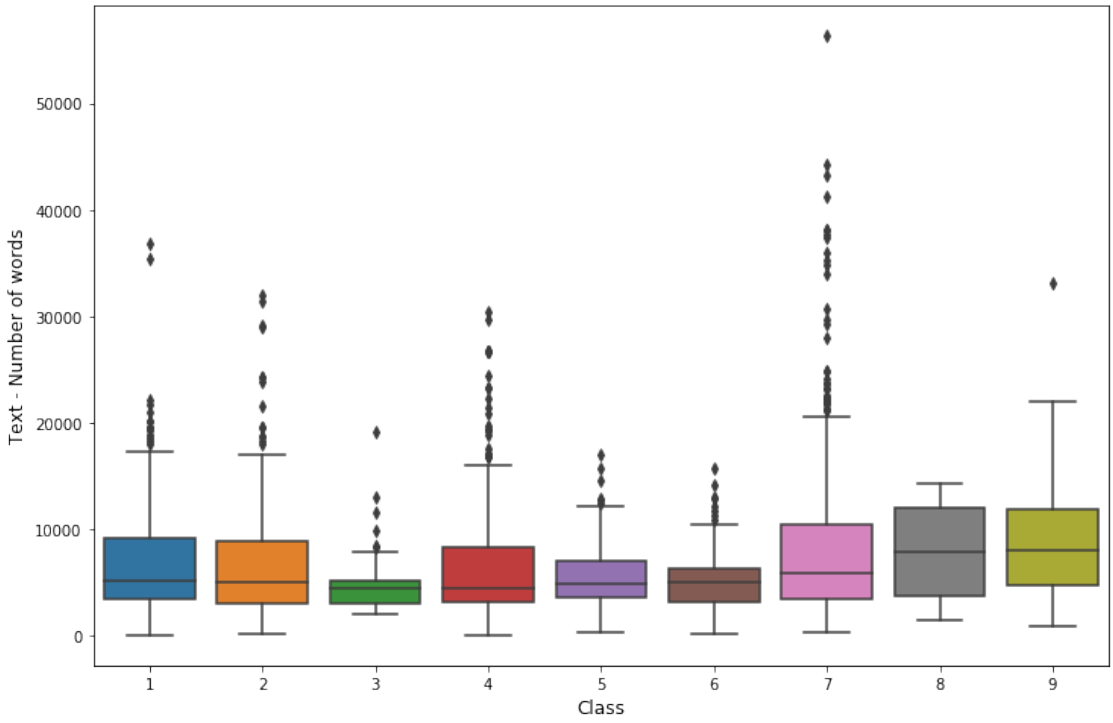


- The distribution is similar to the previous one.

```
In [34]: train_df=pd.merge(x_train,y_train,on="ID",how="left")
```

```
In [35]: plt.figure(figsize=(12,8))
         sns.boxplot(x='Class', y='Text_num_words', data=train_df)
         plt.xlabel('Class', fontsize=12)
         plt.ylabel('Text - Number of words', fontsize=12)
         plt.show()
```

```
<Figure size 864x576 with 0 Axes>
```



- I think this might be useful to discriminate some of the classes like class 3, 6 from others. So might be good to have in the input features.

```
In [42]: # 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 [46]: import math
         #https://stackoverflow.com/a/1602964
         def get_text_responsecoding(df):
             text_feature_responseCoding = np.zeros((df.shape[0],9))
             for i in range(0,9):
                 row_index = 0
                 for index, row in df.iterrows():
                     sum_prob = 0
```

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

In [43]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['TEXT'])


# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [44]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[y_train['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(x_train)


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

In [47]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
```

```
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)
```

In [48]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_fe
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res
```

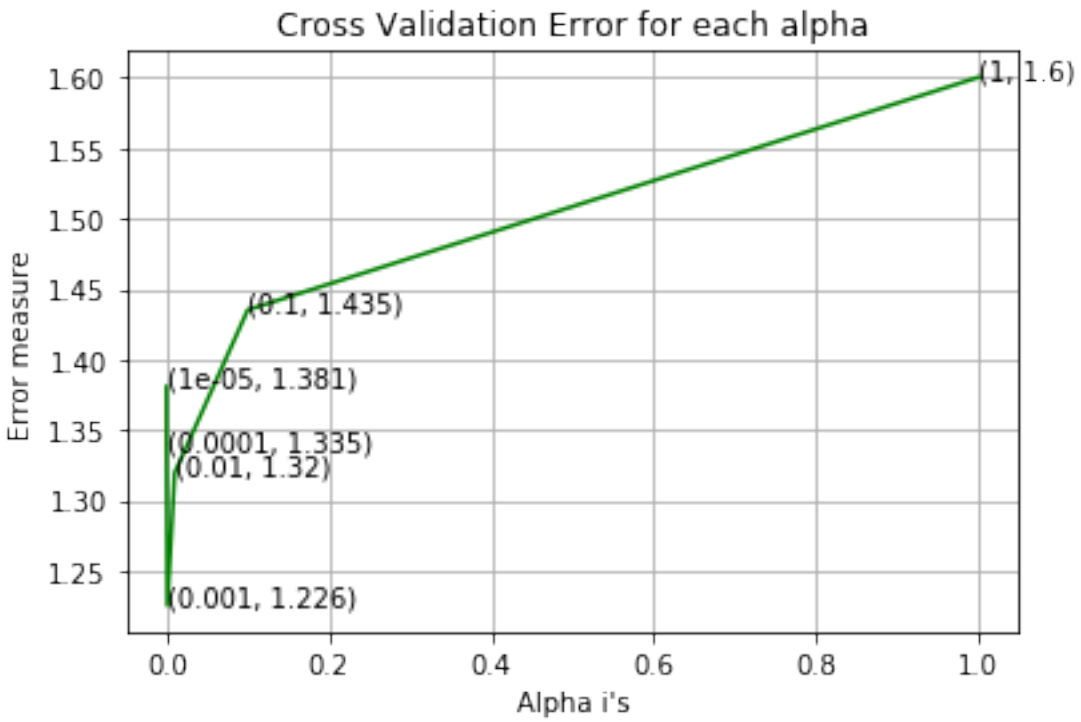
In [49]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [50]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [51]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3808180089563213
For values of alpha =  0.0001 The log loss is: 1.33540071943776
For values of alpha =  0.001 The log loss is: 1.2255925606404456
For values of alpha =  0.01 The log loss is: 1.3195419517231015
For values of alpha =  0.1 The log loss is: 1.435199765285475
For values of alpha =  1 The log loss is: 1.6002248087140585
```




```
For values of best alpha =  0.001 The train log loss is: 0.6889965527448524
For values of best alpha =  0.001 The cross validation log loss is: 1.2255925606404456
For values of best alpha =  0.001 The test log loss is: 1.1364965987862345
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it seems like!

```
In [26]: def get_intersec_text(df):
             df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
             df_text_fea = df_text_vec.fit_transform(df['TEXT'])
             df_text_features = df_text_vec.get_feature_names()

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

In [27]: len1,len2 = get_intersec_text(x_test)
         print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
         len1,len2 = get_intersec_text(x_cv)
         print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train
```

```
94.5 % of word of test data appeared in train data
92.4 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

```
In [51]: #Data preparation for ML models.

         #Misc. functionns for ML models

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

             # for calculating log_loss we willl provide the array of probabilities belongs to
             print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
             # calculating the number of data points that are misclassified
             print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/tes
             plot_confusion_matrix(test_y, pred_y)

In [52]: def report_log_loss(train_x, train_y, test_x, test_y,  clf):
             clf.fit(train_x, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x, train_y)
             sig_clf_probs = sig_clf.predict_proba(test_x)
             return log_loss(test_y, sig_clf_probs, eps=1e-15)

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

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

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

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

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

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

Stacking the three types of features

```
In [54]: # merging gene, variance and text features

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

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

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

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

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


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

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

In [55]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotC
print("(number of data points * number of features) in test data = ", test_x_onehotCod
print("(number of data points * number of features) in cross validation data =", cv_x_
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3189)
(number of data points * number of features) in test data =  (665, 3189)
(number of data points * number of features) in cross validation data = (532, 3189)
```

In [56]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respons
print("(number of data points * number of features) in test data = ", test_x_responseC
print("(number of data points * number of features) in cross validation data =", cv_x_
```

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

4.1. Base Line Model

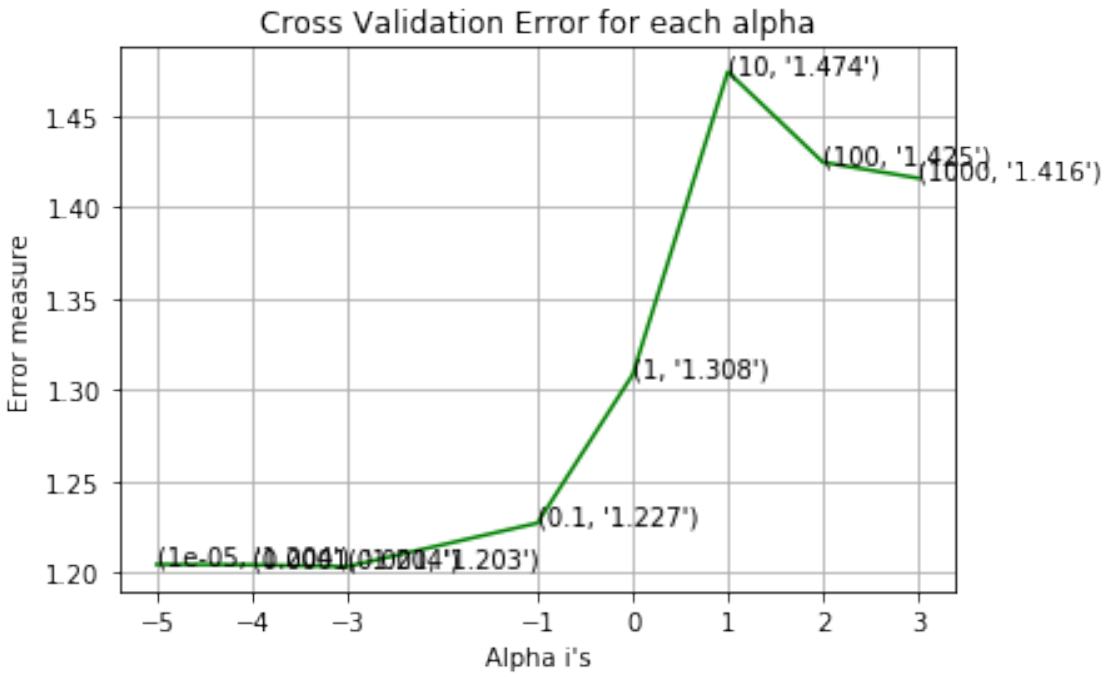
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [57]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.20424862001
for alpha = 0.0001
Log Loss : 1.20398850011
for alpha = 0.001
Log Loss : 1.20302696993
for alpha = 0.1
Log Loss : 1.22692395648
for alpha = 1
Log Loss : 1.30826014838
for alpha = 10
Log Loss : 1.47449178303
for alpha = 100
Log Loss : 1.42480715215
for alpha = 1000
Log Loss : 1.4161173498
```



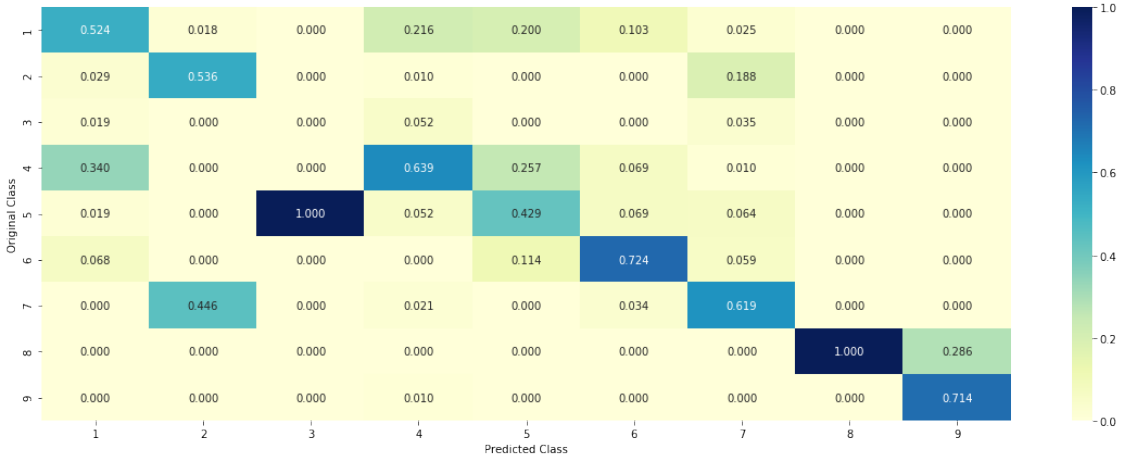

```
For values of best alpha =  0.001 The train log loss is: 0.499965711617
For values of best alpha =  0.001 The cross validation log loss is: 1.20302696993
For values of best alpha =  0.001 The test log loss is: 1.2199021623
```

4.1.1.2. Testing the model with best hyper paramters

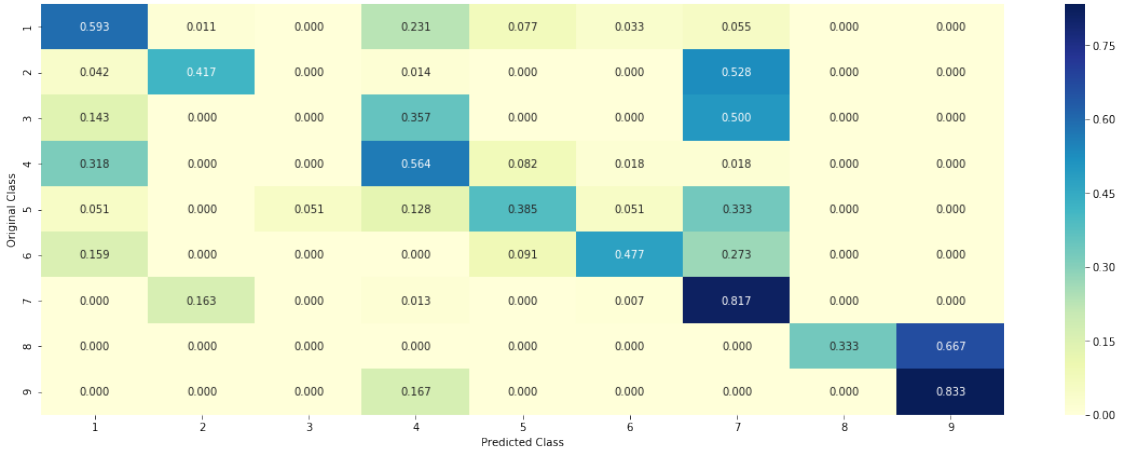
In [58]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.20302696993
Number of missclassified point : 0.4116541353383459

-------------------- Confusion matrix --------------------
```

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



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



4.1.1.3. Feature Importance

```
In [68]: test_point_index = 60
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index],x_test['Gene']
```

```
Predicted Class : 9
Predicted Class Probabilities: [[ 0.0653  0.0446  0.0101  0.0667  0.0356  0.0376  0.087   0.003
Actual Class : 9
--------------------------------------------------
45 Text feature [02] present in test data point [True]
48 Text feature [11] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.2. K Nearest Neighbour Classification

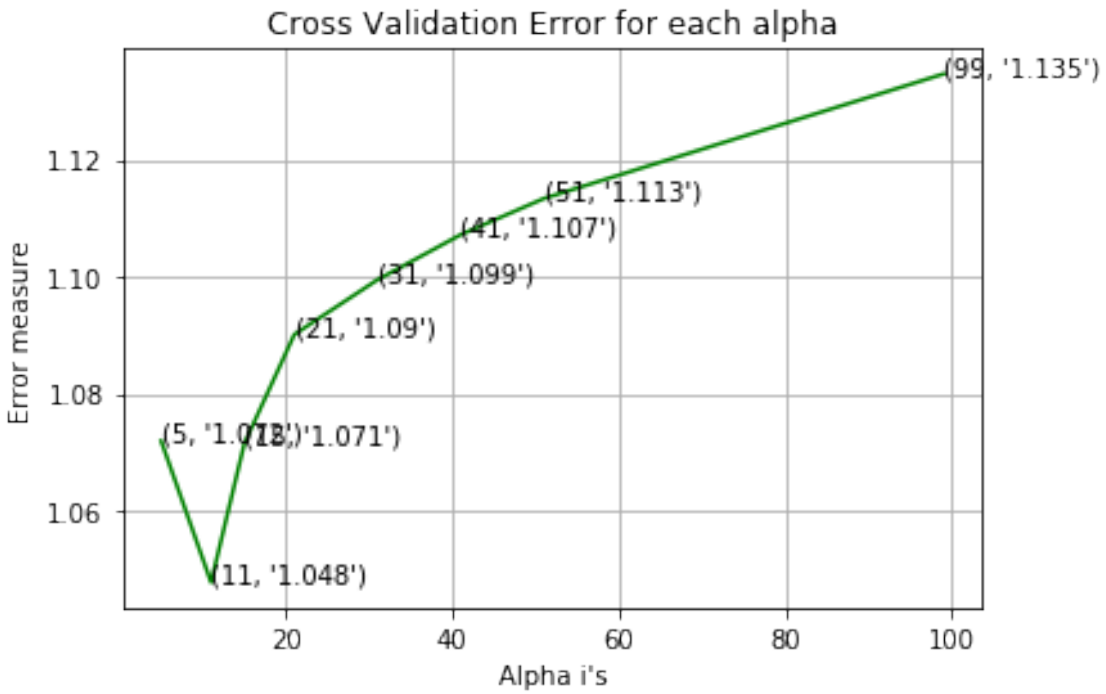
4.2.1. Hyper parameter tuning

```
In [69]: # find more about KNeighborsClassifier()
         # here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsC
         # -------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, p
         # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

         # methods of
         # fit(X, y) : Fit the model using X as training data and y as target values
```

```
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
#--------------------------------------


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


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

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


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
```

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

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

```
for alpha = 5
Log Loss : 1.07190593497
for alpha = 11
Log Loss : 1.04771637946
for alpha = 15
Log Loss : 1.0714588435
for alpha = 21
Log Loss : 1.09009556553
for alpha = 31
Log Loss : 1.09946249182
for alpha = 41
Log Loss : 1.10717241586
for alpha = 51
Log Loss : 1.11336376832
for alpha = 99
Log Loss : 1.13465933939
```



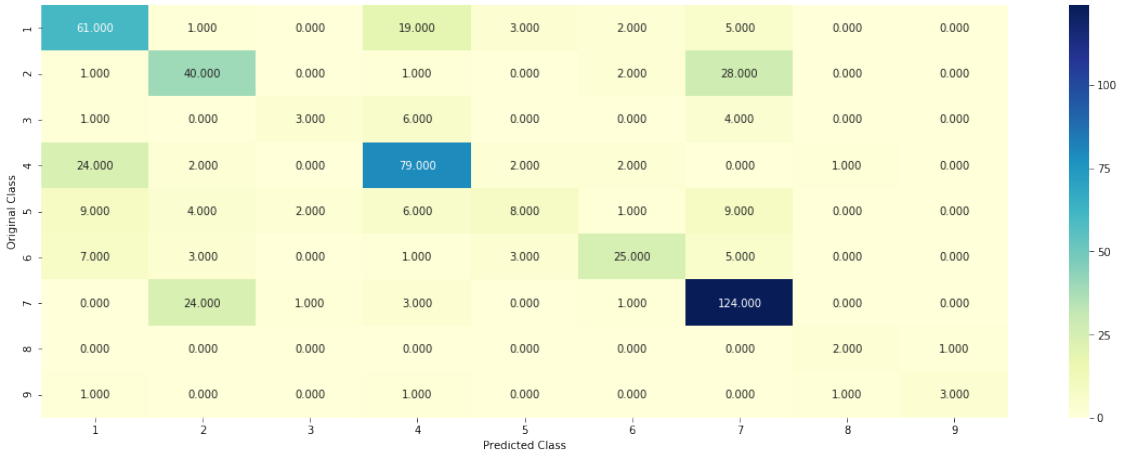

```
For values of best alpha =  11 The train log loss is: 0.63619584131
For values of best alpha =  11 The cross validation log loss is: 1.04771637946
For values of best alpha =  11 The test log loss is: 1.06388675896
```

4.2.2. Testing the model with best hyper paramters

```
In [70]: # find more about KNeighborsClassifier()
         # here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsC
         # -------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
         # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.04771637946
Number of mis-classified points : 0.35150375939849626
```

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



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

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

4.2.3.Sample Query point -1

```
In [71]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 2
Actual Class : 5
The  11  nearest neighbours of the test points belongs to classes [7 3 7 7 5 3 3 2 3 7 7]
Fequency of nearest points : Counter({7: 5, 3: 4, 5: 1, 2: 1})
```

4.2.4. Sample Query Point-2

```
In [72]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 100

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

```
        neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alp
        print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the te
        print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 2
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [7 7
Fequency of nearest points : Counter({7: 7, 2: 2, 6: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

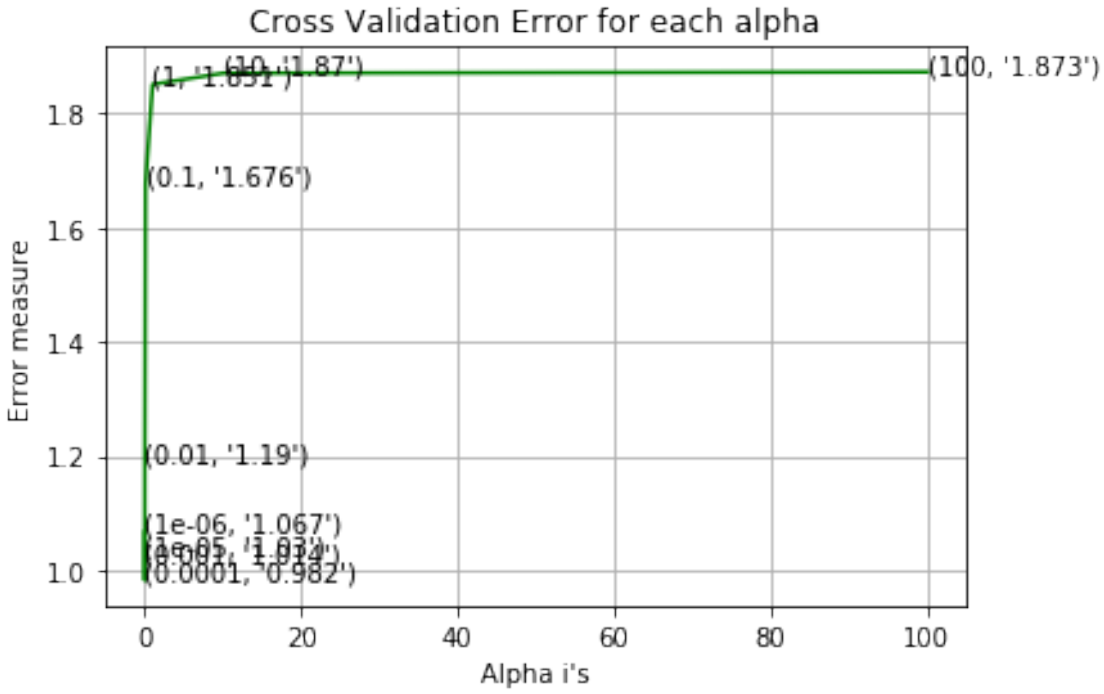
4.3.1.1. Hyper paramter tuning

```
In [73]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

         alpha = [10 ** x for x in range(-6, 3)]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
```

```
        clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
        # to avoid rounding error while multiplying probabilites we use log-probability e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


    best_alpha = np.argmin(cv_log_error_array)
    clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)

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

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

```
for alpha = 1e-06
Log Loss : 1.06734708683
for alpha = 1e-05
Log Loss : 1.02970064266
```

```
for alpha = 0.0001
Log Loss : 0.982464068085
for alpha = 0.001
Log Loss : 1.01351787532
for alpha = 0.01
Log Loss : 1.1903441633
for alpha = 0.1
Log Loss : 1.67642776227
for alpha = 1
Log Loss : 1.85094765201
for alpha = 10
Log Loss : 1.87044338695
for alpha = 100
Log Loss : 1.8727230987
```



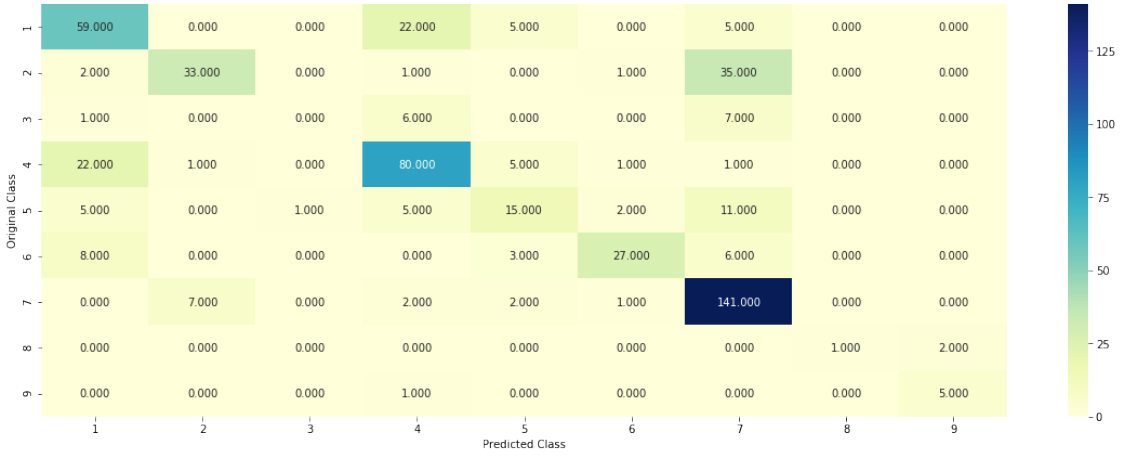

```
For values of best alpha =  0.0001 The train log loss is: 0.428647067051
For values of best alpha =  0.0001 The cross validation log loss is: 0.982464068085
For values of best alpha =  0.0001 The test log loss is: 1.02838376874
```

4.3.1.2. Testing the model with best hyper paramters

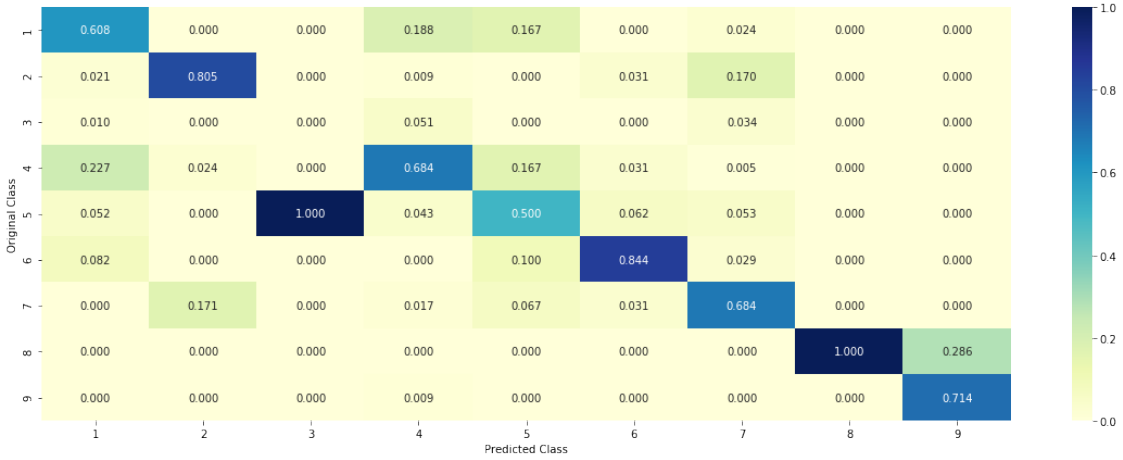
```
In [74]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
```

```
        # default parameters
        # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
        # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
        # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

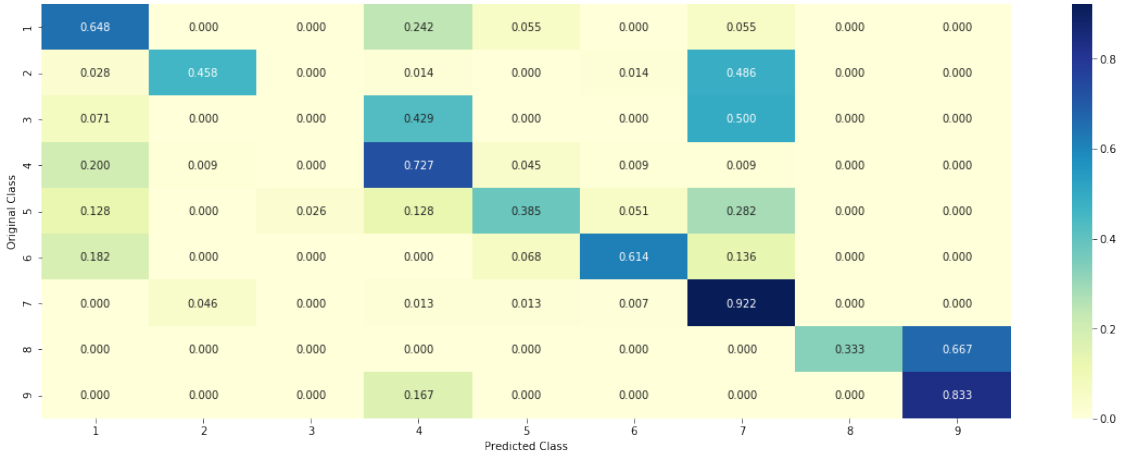
```
Log loss : 0.982464068085
Number of mis-classified points : 0.32142857142857145

-------------------- Confusion matrix --------------------
```




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



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



4.3.1.3. Feature Importance

```
In [75]: def get_imp_feature_names(text, indices, removed_ind = []):
             word_present = 0
             tabulte_list = []
             incresingorder_ind = 0
             for i in indices:
                 if i < train_gene_feature_onehotCoding.shape[1]:
                     tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                 elif i< 18:
                     tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                 if ((i > 17) & (i not in removed_ind)) :
                     word = train_text_features[i]
```

```
                    yes_no = True if word in text.split() else False
                    if yes_no:
                        word_present += 1
                    tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
                incresingorder_ind += 1
            print(word_present, "most importent features are present in our query point")
            print("-"*50)
            print("The features that are most importent of the ",predicted_cls[0]," class:")
            print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Feature Importance

```
In [79]: # from tabulate import tabulate
         clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
         clf.fit(train_x_onehotCoding,train_y)
         test_point_index = 200
         no_feature = 1000
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0],
                              x_test['TEXT'].iloc[test_point_index],
                              x_test['Gene'].iloc[test_point_index],
                              x_test['Variation'].iloc[test_point_index],
                              no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1132  0.0734  0.0053  0.1916  0.0221  0.0247  0.5625  0.004
Actual Class : 5
--------------------------------------------------
7 Text feature [04] present in test data point [True]
95 Text feature [114] present in test data point [True]
185 Text feature [003] present in test data point [True]
640 Text feature [119] present in test data point [True]
854 Text feature [12] present in test data point [True]
982 Text feature [001] present in test data point [True]
Out of the top  1000  features  6 are present in query point
```

4.3.2. Without Class balancing

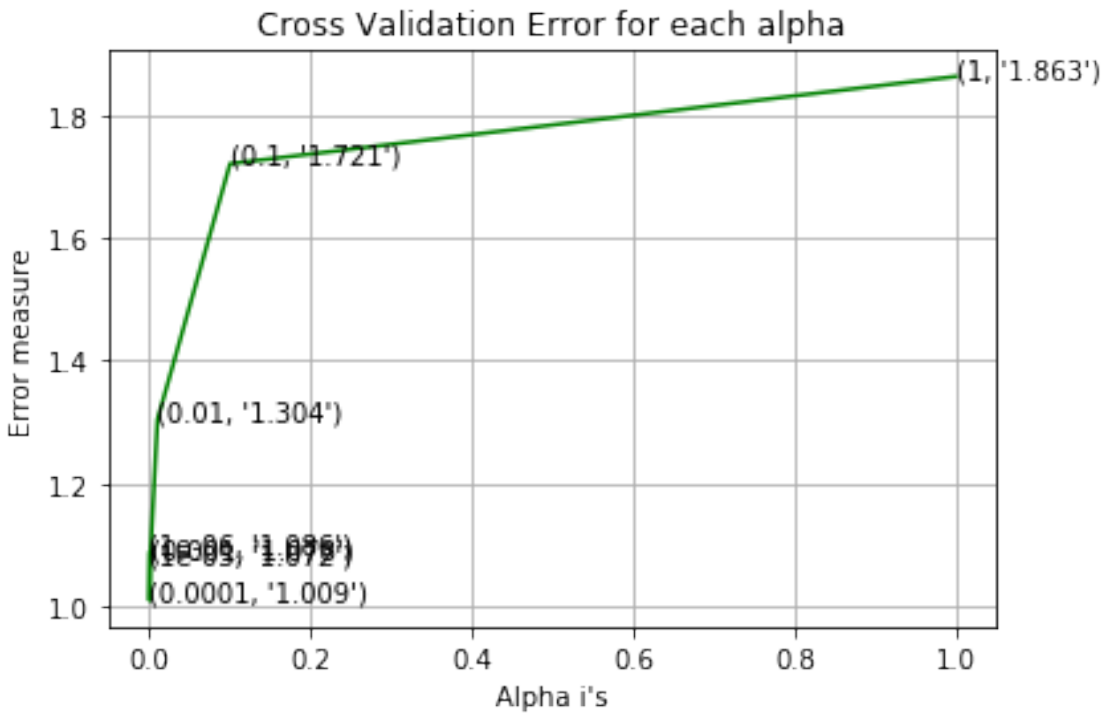
4.3.2.1. Hyper paramter tuning

```
In [80]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.08582165705
for alpha = 1e-05
Log Loss : 1.07245026376
for alpha = 0.0001
Log Loss : 1.00879921873
for alpha = 0.001
Log Loss : 1.07797407912
for alpha = 0.01
Log Loss : 1.30363887102
for alpha = 0.1
Log Loss : 1.7206877331
for alpha = 1
Log Loss : 1.86285538105
```

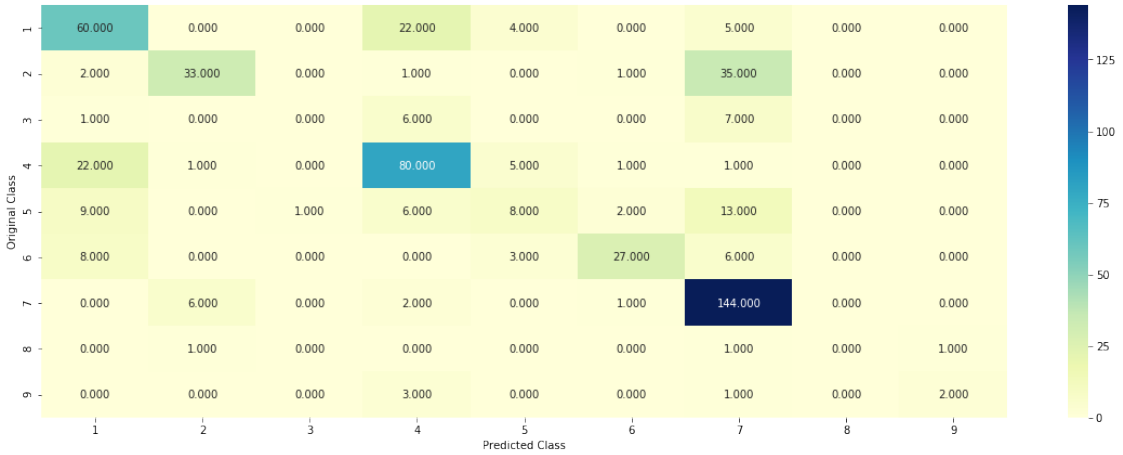



```
For values of best alpha =  0.0001 The train log loss is: 0.419422181369
For values of best alpha =  0.0001 The cross validation log loss is: 1.00879921873
For values of best alpha =  0.0001 The test log loss is: 1.05538771359
```

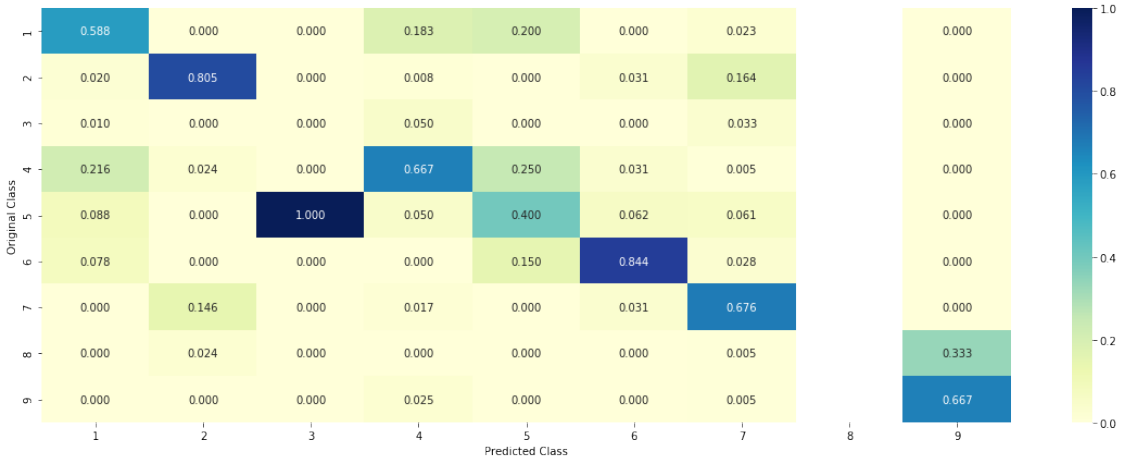
4.3.2.2. Testing model with best hyper parameters

```
In [81]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

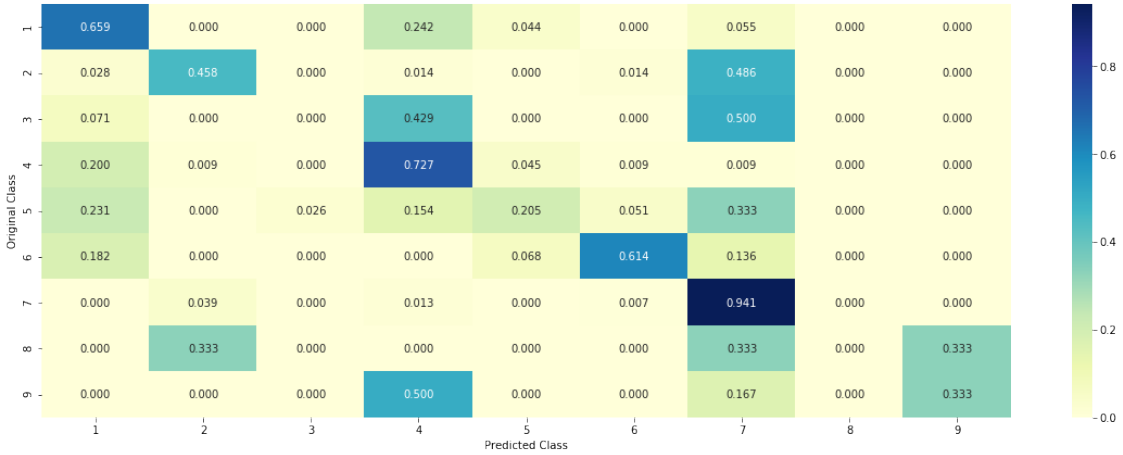
```
Log loss : 1.00879921873
Number of mis-classified points : 0.33458646616541354

-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Feature Importance

```
In [83]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
         clf.fit(train_x_onehotCoding,train_y)
         test_point_index = 55
         no_feature = 1000
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0],
                              x_test['TEXT'].iloc[test_point_index],
                              x_test['Gene'].iloc[test_point_index],
                              x_test['Variation'].iloc[test_point_index],
                              no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[  1.27500000e-01   6.10000000e-03   1.00000000e-03   3.800000
    1.98600000e-01   6.58900000e-03   3.00000000e-03   9.00000000e-04
    1.00000000e-04]]
Actual Class : 6
--------------------------------------------------
142 Text feature [10] present in test data point [True]
148 Text feature [000] present in test data point [True]
302 Text feature [001] present in test data point [True]
376 Text feature [07] present in test data point [True]
416 Text feature [12] present in test data point [True]
522 Text feature [02] present in test data point [True]
Out of the top  1000  features  6 are present in query point
```

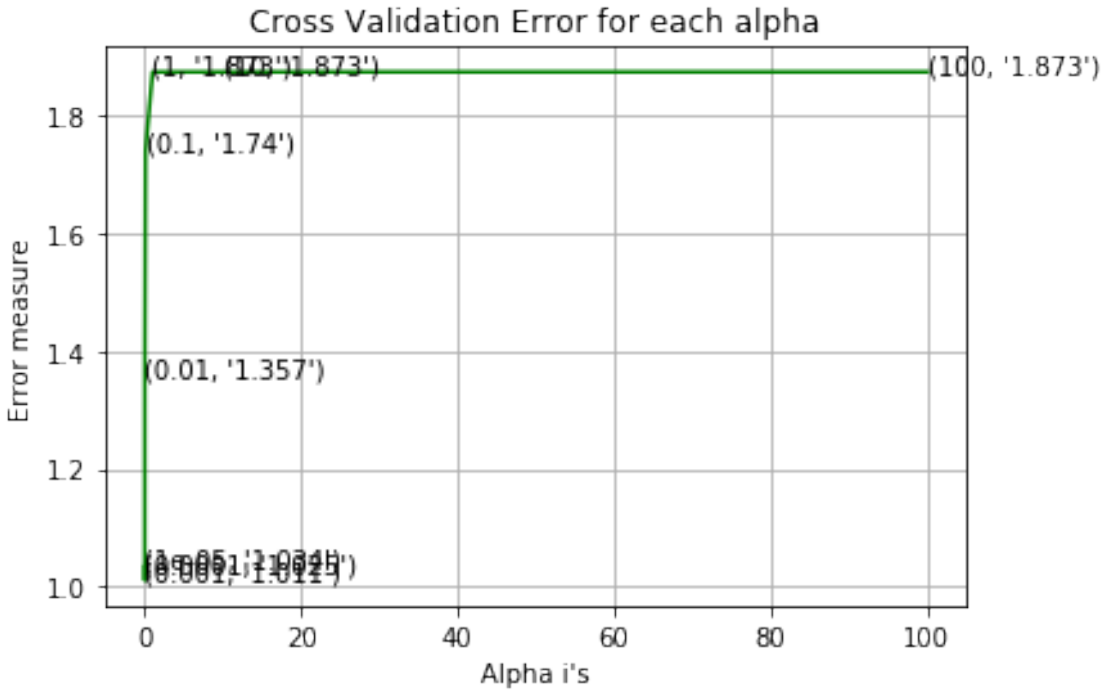
4.4. Linear Support Vector Machines
4.4.1. Hyper paramter tuning

In [84]:
```
# read more about support vector machines with linear kernals here http://scikit-lear

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

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



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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

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


        best_alpha = np.argmin(cv_log_error_array)
        # clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
        clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)

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

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

```
for C = 1e-05
Log Loss : 1.03422373549
for C = 0.0001
Log Loss : 1.02498821204
for C = 0.001
Log Loss : 1.01091459691
for C = 0.01
Log Loss : 1.35732274465
for C = 0.1
Log Loss : 1.73954661087
for C = 1
Log Loss : 1.87332436298
for C = 10
Log Loss : 1.8733238446
```

```
for C = 100
Log Loss : 1.8733240052
```




```
For values of best alpha =  0.001 The train log loss is: 0.577515180822
For values of best alpha =  0.001 The cross validation log loss is: 1.01091459691
For values of best alpha =  0.001 The test log loss is: 1.08547688584
```

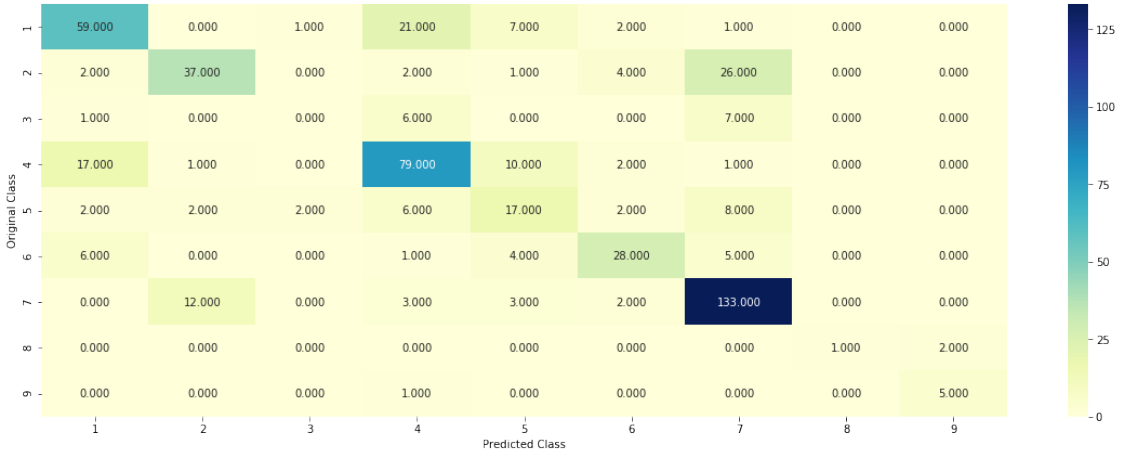
4.4.2. Testing model with best hyper parameters

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

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

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

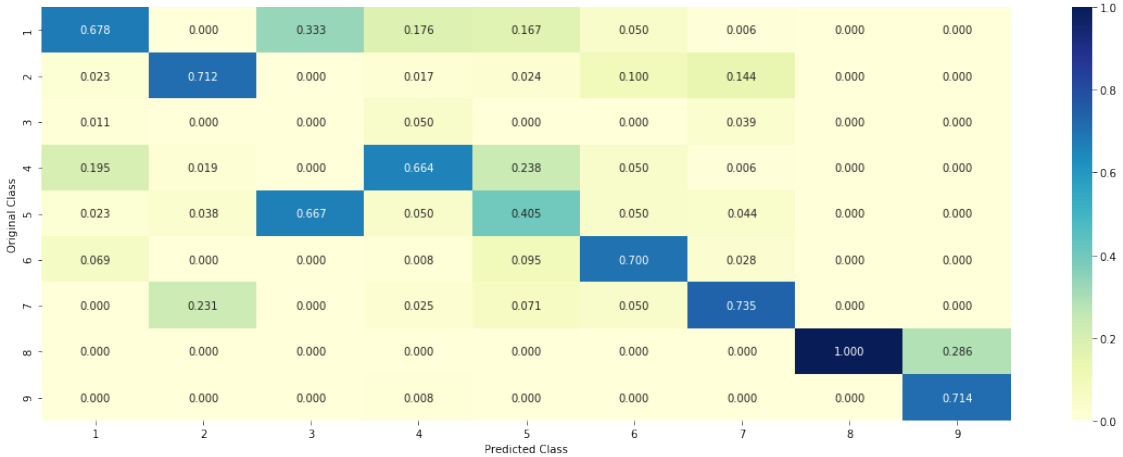
```
    # clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balan
    clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
    predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y
```

```
Log loss : 1.01091459691
Number of mis-classified points : 0.325187969924812
```

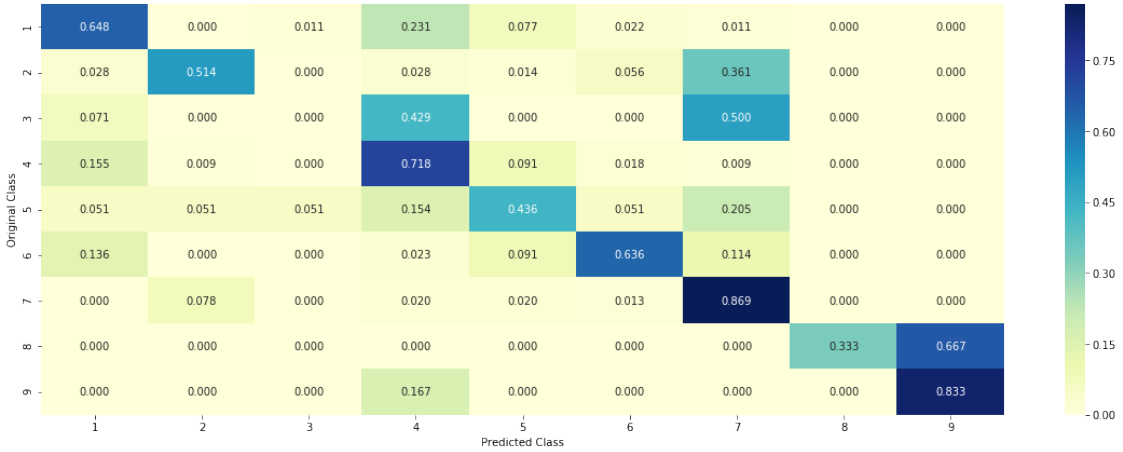
```
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

```
In [86]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
         clf.fit(train_x_onehotCoding,train_y)
         test_point_index = 100
         # test_point_index = 100
         no_feature = 1000
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0],
                              x_test['TEXT'].iloc[test_point_index],
                              x_test['Gene'].iloc[test_point_index],
                              x_test['Variation'].iloc[test_point_index],
                              no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0728  0.086   0.0043  0.0764  0.041   0.0339  0.6742  0.00
Actual Class : 2
--------------------------------------------------
656 Text feature [01] present in test data point [True]
793 Text feature [12] present in test data point [True]
972 Text feature [125] present in test data point [True]
Out of the top  1000  features  3 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

```
In [87]: # --------------------------------
         # default parameters
```

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

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

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

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


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

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

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

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

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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.28175207666
for n_estimators = 100 and max depth =  10
Log Loss : 1.27934799801
for n_estimators = 200 and max depth =  5
Log Loss : 1.25462395371
for n_estimators = 200 and max depth =  10
Log Loss : 1.26464169056
for n_estimators = 500 and max depth =  5
Log Loss : 1.24802689716
for n_estimators = 500 and max depth =  10
Log Loss : 1.26221342239
for n_estimators = 1000 and max depth =  5
```

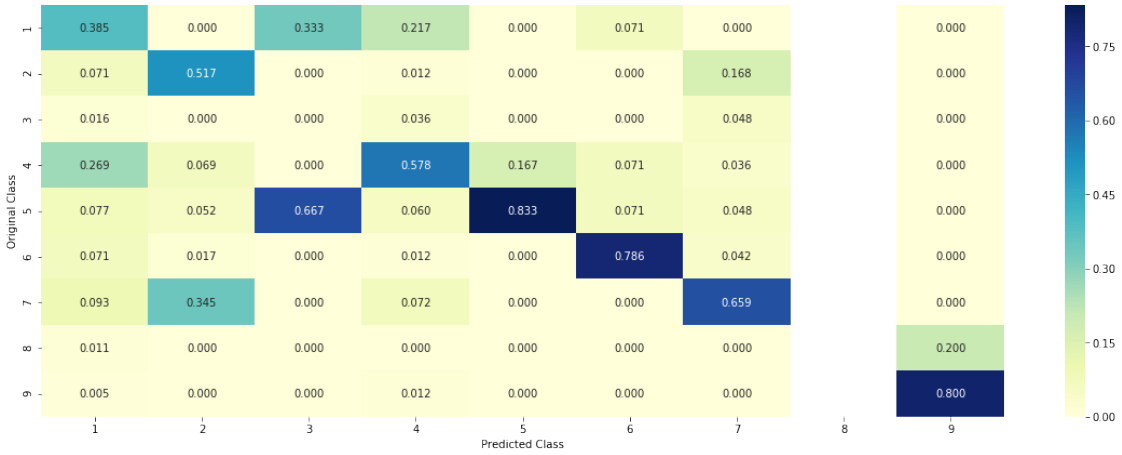
```
Log Loss : 1.24925954395
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2600737394
for n_estimators = 2000 and max depth =  5
Log Loss : 1.25133318188
for n_estimators = 2000 and max depth =  10
Log Loss : 1.26396470151
For values of best estimator =  500 The train log loss is: 0.879746374326
For values of best estimator =  500 The cross validation log loss is: 1.24802689716
For values of best estimator =  500 The test log loss is: 1.2426565132
```

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

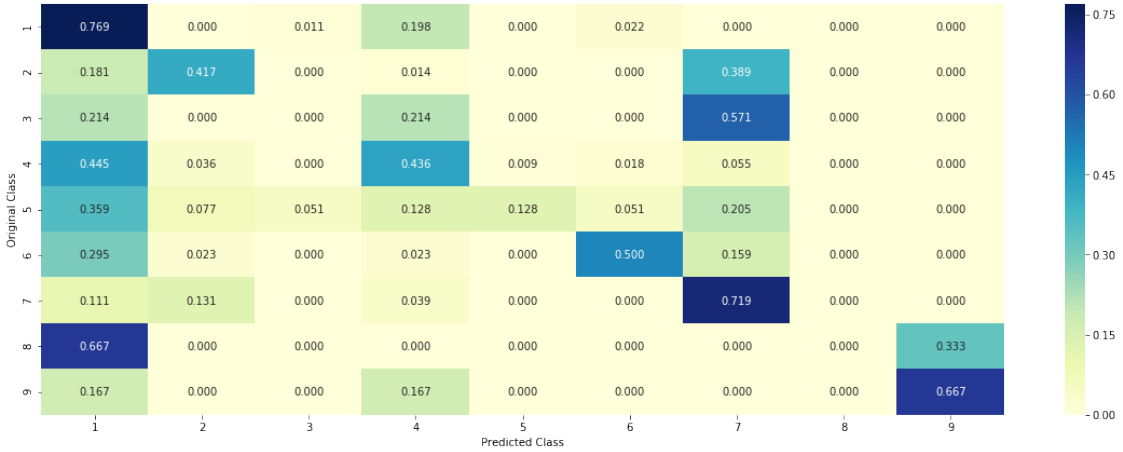
```
In [88]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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

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

```
Log loss : 1.24802689716
Number of mis-classified points : 0.4567669172932331

-------------------- Confusion matrix --------------------
```

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



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



4.5.3. Feature Importance

```
In [89]: # test_point_index = 10
         clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
         clf.fit(train_x_onehotCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_onehotCoding, train_y)

         test_point_index = 10
         no_feature = 1000
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         get_impfeature_names(indices[:no_feature],
                              x_test['TEXT'].iloc[test_point_index],
                              x_test['Gene'].iloc[test_point_index],
                              x_test['Variation'].iloc[test_point_index],
                              no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.3529  0.1304  0.008   0.0876  0.0506  0.0488  0.1225  0.030
Actual Class : 4
--------------------------------------------------
16 Text feature [1211] present in test data point [True]
29 Text feature [108] present in test data point [True]
94 Text feature [13] present in test data point [True]
101 Text feature [003] present in test data point [True]
109 Text feature [10] present in test data point [True]
123 Text feature [104] present in test data point [True]
```

```
161 Text feature [12c] present in test data point [True]
172 Text feature [1053] present in test data point [True]
190 Text feature [105] present in test data point [True]
232 Text feature [11] present in test data point [True]
316 Text feature [102] present in test data point [True]
319 Text feature [000] present in test data point [True]
321 Text feature [1000] present in test data point [True]
324 Text feature [001] present in test data point [True]
327 Text feature [1018] present in test data point [True]
332 Text feature [007] present in test data point [True]
358 Text feature [05] present in test data point [True]
375 Text feature [107] present in test data point [True]
387 Text feature [002] present in test data point [True]
444 Text feature [131] present in test data point [True]
518 Text feature [02] present in test data point [True]
528 Text feature [115] present in test data point [True]
554 Text feature [017] present in test data point [True]
562 Text feature [103] present in test data point [True]
598 Text feature [018] present in test data point [True]
627 Text feature [069] present in test data point [True]
675 Text feature [12a] present in test data point [True]
698 Text feature [03] present in test data point [True]
723 Text feature [118] present in test data point [True]
725 Text feature [12] present in test data point [True]
727 Text feature [082] present in test data point [True]
732 Text feature [130] present in test data point [True]
753 Text feature [032] present in test data point [True]
757 Text feature [022] present in test data point [True]
760 Text feature [12463] present in test data point [True]
834 Text feature [132] present in test data point [True]
862 Text feature [040] present in test data point [True]
873 Text feature [01] present in test data point [True]
903 Text feature [019] present in test data point [True]
908 Text feature [125] present in test data point [True]
928 Text feature [029] present in test data point [True]
950 Text feature [1011] present in test data point [True]
954 Text feature [12b] present in test data point [True]
968 Text feature [100] present in test data point [True]
Out of the top  1000  features  44 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

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

```
# class_weight=None)

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

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

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


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

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

```
            ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_e
        plt.grid()
        plt.title("Cross Validation Error for each alpha")
        plt.xlabel("Alpha i's")
        plt.ylabel("Error measure")
        plt.show()
        '''

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

        predict_y = sig_clf.predict_proba(train_x_responseCoding)
        print('For values of best alpha = ',
              alpha[int(best_alpha/4)],
              "The train log loss is:",
              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 cross validation log loss is:",
              log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

        predict_y = sig_clf.predict_proba(test_x_responseCoding)
        print('For values of best alpha = ',
              alpha[int(best_alpha/4)],
              "The test log loss is:",
              log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  5
Log Loss : 1.50229855863
for n_estimators = 10 and max depth =  10
Log Loss : 1.89747007886
for n_estimators = 50 and max depth =  5
Log Loss : 1.38649115516
for n_estimators = 50 and max depth =  10
Log Loss : 1.70029670426
for n_estimators = 100 and max depth =  5
Log Loss : 1.29329848058
for n_estimators = 100 and max depth =  10
Log Loss : 1.65355848728
for n_estimators = 200 and max depth =  5
Log Loss : 1.30850829019
for n_estimators = 200 and max depth =  10
Log Loss : 1.70433305795
```

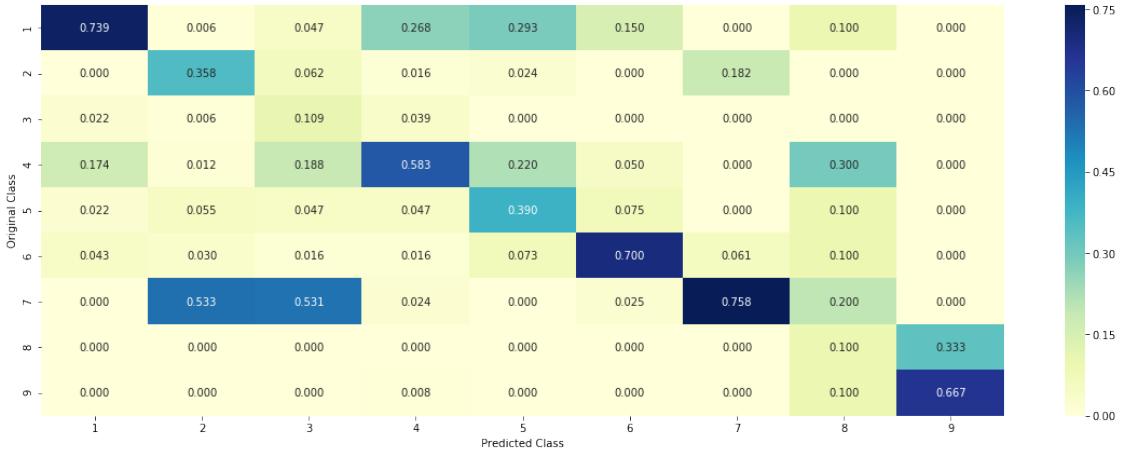
```
For values of best alpha =  50 The train log loss is: 0.0645169263409
For values of best alpha =  50 The cross validation log loss is: 1.38649115516
For values of best alpha =  50 The test log loss is: 1.38910415007
```

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

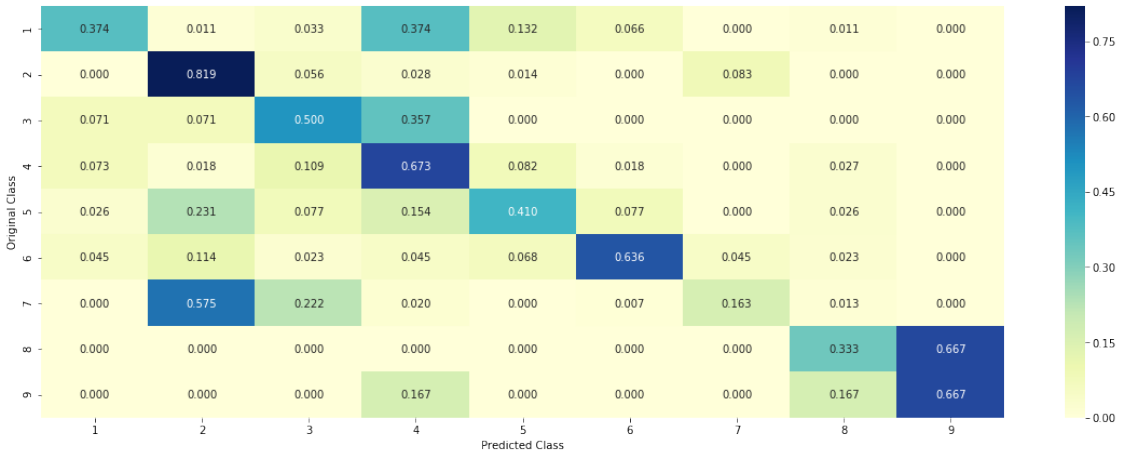
```
In [91]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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

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

```
Log loss : 1.38649115516
Number of mis-classified points : 0.5338345864661654

-------------------- Confusion matrix --------------------
```

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




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



4.5.5. Feature Importance

```
In [92]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)
```

```
        test_point_index = 100
        no_feature = 1000
        predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_response
        print("Actual Class :", test_y[test_point_index])
        # indices = np.argsort(-clf.feature_importances_)
        # print("-"*50)
        # for i in indices:
        #     if i<9:
        #         print("Gene is important feature")
        #     elif i<18:
        #         print("Variation is important feature")
        #     else:
        #         print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0166  0.342   0.2085  0.0232  0.0319  0.0438  0.305   0.018
Actual Class : 2
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```
In [93]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

         # Some of methods of SVM()
```

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


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

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

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

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


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

clf2 = SGDClassifier(alpha=0.01, penalty='l2', loss='hinge', class_weight='balanced',
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=1000)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_prob
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_p
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_or
```

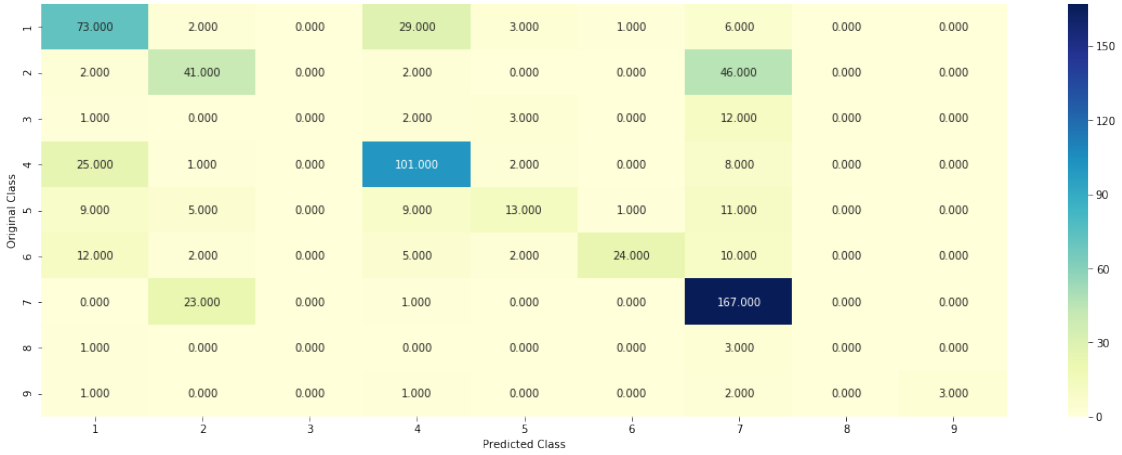
```
        print("-"*50)
        alpha = [0.0001,0.001,0.01,0.1,1,10]
        best_alpha = 999
        for i in alpha:
            lr = LogisticRegression(C=i)
            sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classi
            sclf.fit(train_x_onehotCoding, train_y)
            print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_
            log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
            if best_alpha > log_error:
                best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.02
Support vector machines : Log Loss: 1.35
Naive Bayes : Log Loss: 1.42
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.503
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.094
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.090
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.278
```

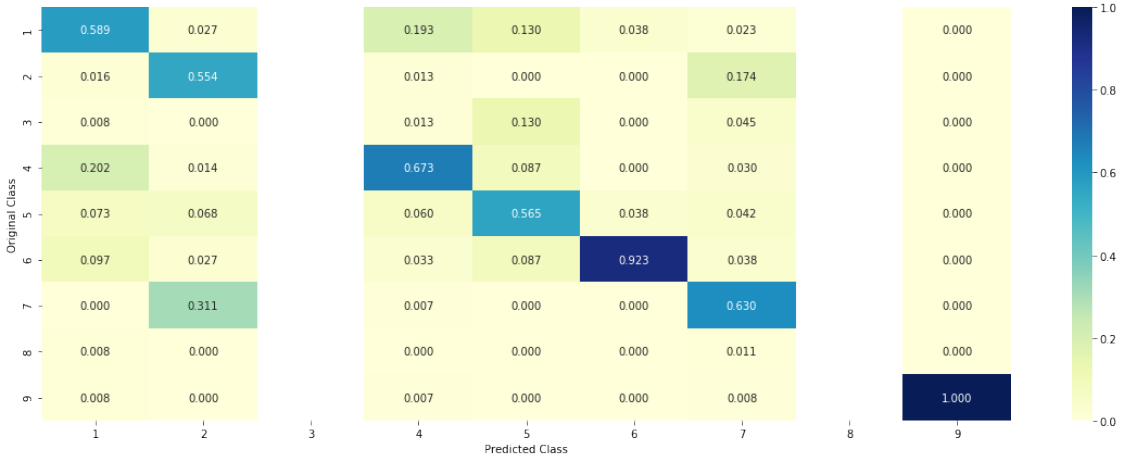
4.7.2 testing the model with the best hyper parameters

```
In [94]: lr = LogisticRegression(C=0.1)
         sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=
         sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

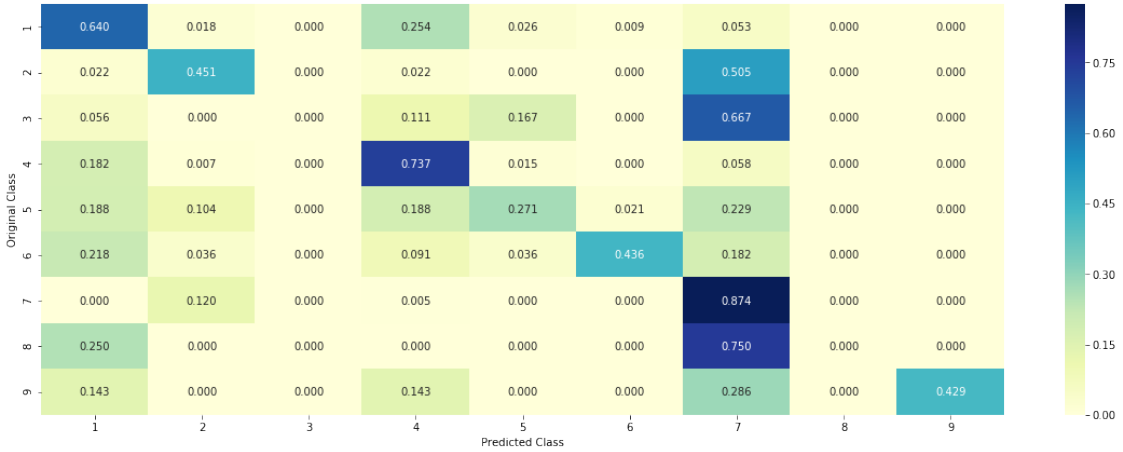
```
Log loss (train) on the stacking classifier : 0.89544626715
Log loss (CV) on the stacking classifier : 1.09395267034
Log loss (test) on the stacking classifier : 1.14428242187
Number of missclassified point : 0.36541353383458647

-------------------- Confusion matrix --------------------
```



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



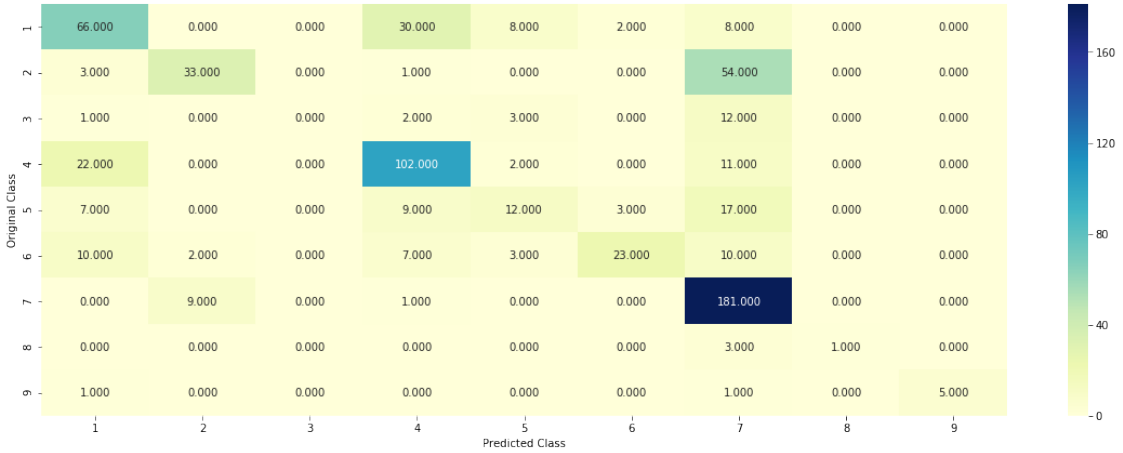
---------- Recall matrix (Row sum=1) ----------



4.7.3 Maximum Voting classifier

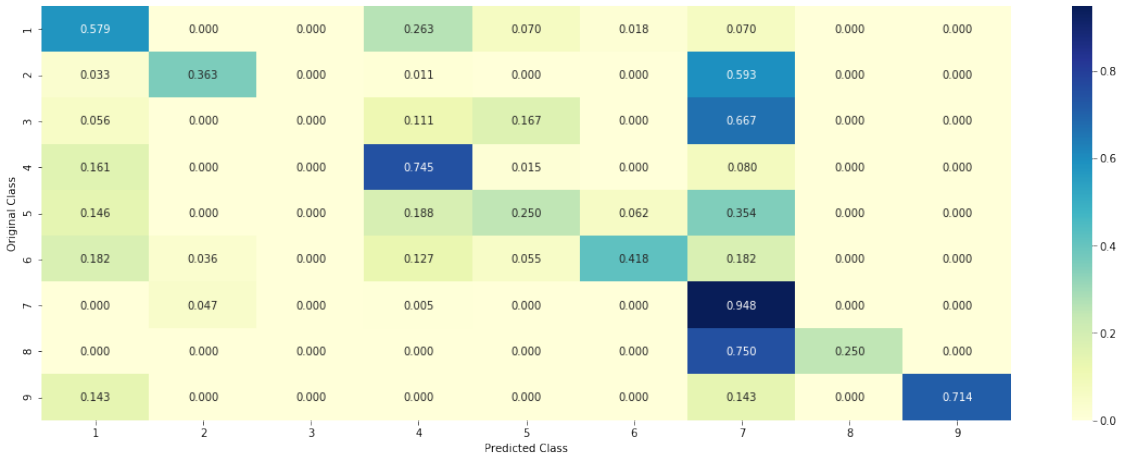
```
In [95]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassi
         from sklearn.ensemble import VotingClassifier
         vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_cl
         vclf.fit(train_x_onehotCoding, train_y)
         print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_pr
         print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv
         print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba
         print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_oneho
         plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.940804444632
Log loss (CV) on the VotingClassifier : 1.11994390624
Log loss (test) on the VotingClassifier : 1.1628453832
Number of missclassified point : 0.36390977443609024

---------- Confusion matrix ----------
```



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


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



Logistic Regression With Class Balancing
Gene Feature

```
In [127]: #response-coding of the Gene feature
          # alpha is used for laplace smoothing
          alpha = 1
```

```
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_train))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_test))

# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_cv))
```

In [128]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])

# don't forget to normalize every feature
train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCoding, axis=0)
test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCoding, axis=0)
cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding, axis=0)
```

Variation Feature

In [129]:
```
# alpha is used for laplace smoothing
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_c
```

In [130]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['Va
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variatio
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'])


# don't forget to normalize every feature
train_variation_feature_onehotCoding = normalize(train_variation_feature_onehotCoding
test_variation_feature_onehotCoding = normalize(test_variation_feature_onehotCoding,
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_onehotCoding, axis
```

Text Feature

In [131]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1, 2))
```

```
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['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*n
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 772782
```

In [132]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# 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_
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feat
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_re
```

In [133]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

Stack above three features

In [135]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
```

```
#       [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variatio
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fe
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding))
cv_y = np.array(list(y_cv['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_va
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_varia
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_f

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_re
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_response
```

In [136]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehot
print("(number of data points * number of features) in test data = ", test_x_onehotCo
print("(number of data points * number of features) in cross validation data =", cv_x
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 775087)
(number of data points * number of features) in test data =  (665, 775087)
(number of data points * number of features) in cross validation data = (532, 775087)
```

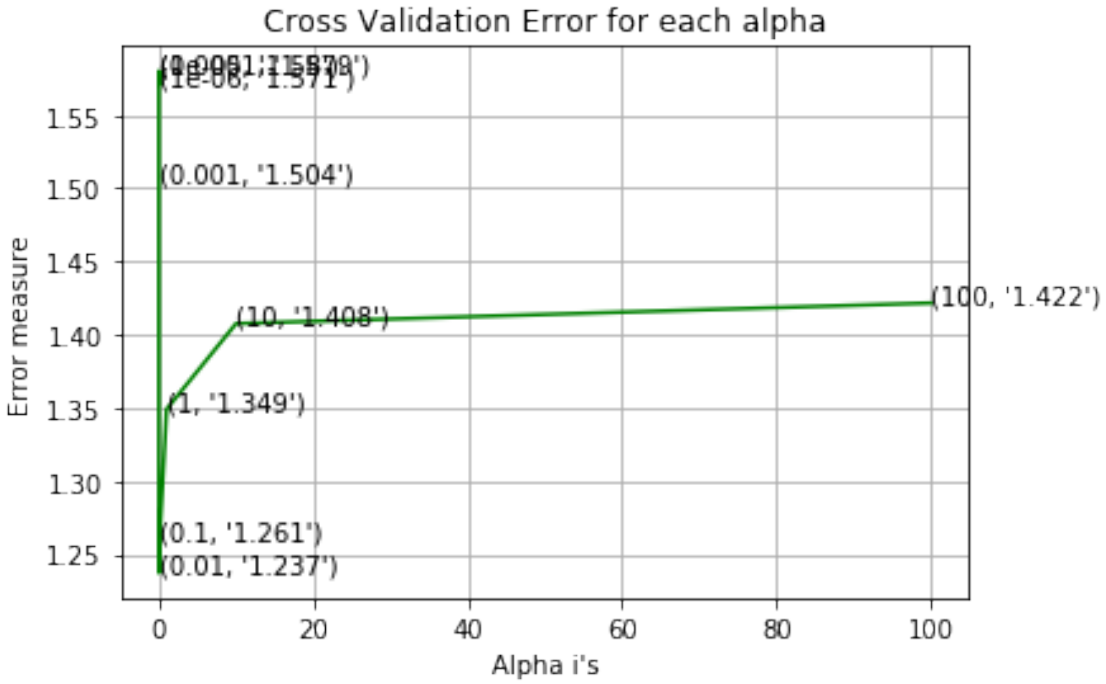
In [137]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respon
print("(number of data points * number of features) in test data = ", test_x_response
print("(number of data points * number of features) in cross validation data =", cv_x
```

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

Lets apply Logistic Regression

```
In [138]: 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', r
              clf.fit(train_x_onehotCoding, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_onehotCoding, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
              cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
              # to avoid rounding error while multiplying probabilites we use log-probability
              print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


          best_alpha = np.argmin(cv_log_error_array)
          clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
          clf.fit(train_x_onehotCoding, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_onehotCoding, train_y)

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

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

for alpha = 1e-06
```

```
Log Loss : 1.5714480791793959
for alpha = 1e-05
Log Loss : 1.5802457581271678
for alpha = 0.0001
Log Loss : 1.5786920609970978
for alpha = 0.001
Log Loss : 1.5043825967185898
for alpha = 0.01
Log Loss : 1.2372499203303868
for alpha = 0.1
Log Loss : 1.260997045892501
for alpha = 1
Log Loss : 1.3490610559798297
for alpha = 10
Log Loss : 1.407595926210261
for alpha = 100
Log Loss : 1.4216165080161391
```

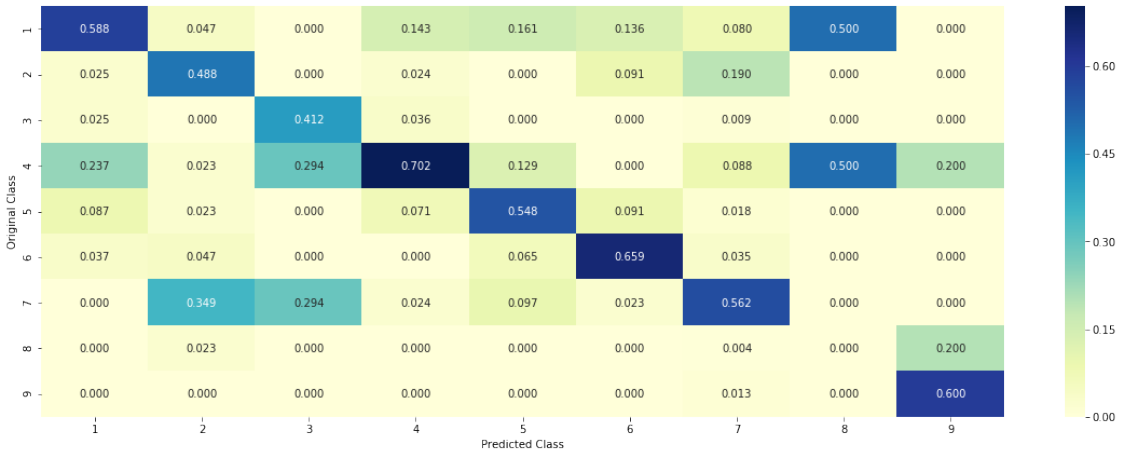



```
For values of best alpha =  0.01 The train log loss is: 0.8776692573531933
For values of best alpha =  0.01 The cross validation log loss is: 1.2372499203303868
For values of best alpha =  0.01 The test log loss is: 1.192856481504718
```

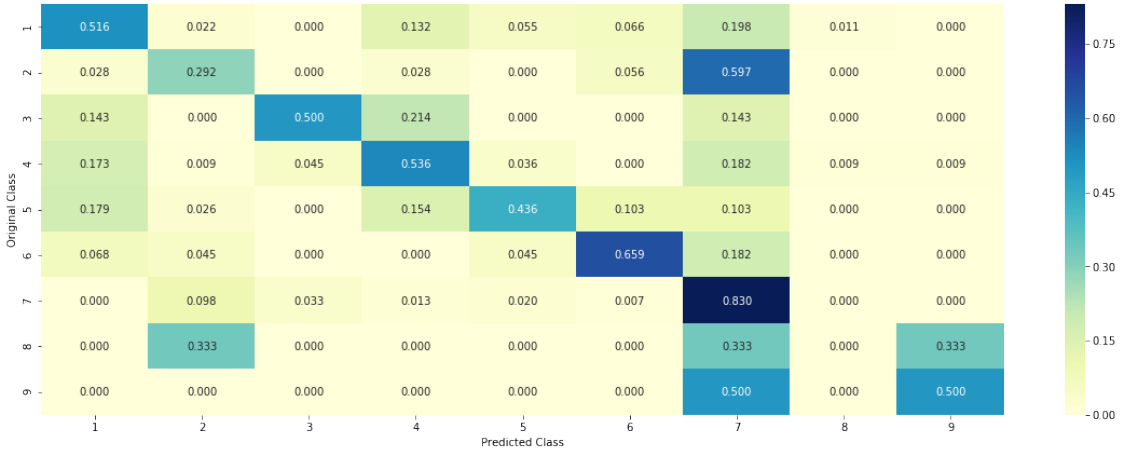
```
In [139]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
          predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, c
```

```
Log loss : 1.2372499203303868
Number of mis-classified points : 0.41729323308270677

-------------------- Confusion matrix --------------------
```


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




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



Here model can't decreases log loss values after using unigram and bigram features

Now we will do some feature engineering(like merging the columns together) on the data and then apply logistic regression again

Gene Feature

```
In [97]: result = pd.merge(data_variants, data_text,on='ID', how='left')
         result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
         y_true = result['Class'].values
         result.Gene = result.Gene.str.replace('\s+', '_')
         result.Variation = result.Variation.str.replace('\s+', '_')

         x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, t

         x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train, te
```

```
In [98]: # get_gv_fea_dict: Get Gene varaition Feature Dict
         def get_gv_fea_dict(alpha, feature, df):
             value_count = x_train[feature].value_counts()
             gv_dict = dict()
             for i, denominator in value_count.items():
                 vec = []
                 for k in range(1,10):
                     cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
                     vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
                 gv_dict[i]=vec
             return gv_dict

         # Get Gene variation feature
         def get_gv_feature(alpha, feature, df):
             gv_dict = get_gv_fea_dict(alpha, feature, df)
             value_count = x_train[feature].value_counts()
             gv_fea = []
```

```
            for index, row in df.iterrows():
                if row[feature] in dict(value_count).keys():
                    gv_fea.append(gv_dict[row[feature]])
                else:
                    gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
            return gv_fea
```

In [99]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1

# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_train))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_test))

# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_cv))
```

In [100]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

Variation Feature

In [101]:
```
# alpha is used for laplace smoothing
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_c
```

In [102]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['Va
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variatio
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'])
```

Text Feature

In [103]:
```
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


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

In [104]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in trai
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['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*n
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 112387
```

In [105]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['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(x_train)


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

In [106]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# 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_
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feat
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_re
```

In [107]:
```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['TEXT'])
```

Features after feature engineering

In [108]:
```
# Collecting all the genes and variations data into a single list
gene_variation = []

for gene in data_variants['Gene'].values:
    gene_variation.append(gene)

for variation in data_variants['Variation'].values:
    gene_variation.append(variation)
```

In [109]:
```
tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
cv_text = tfidfVectorizer.transform(x_cv['TEXT'])
```

Stack above three features

In [110]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variatio
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fe
```

```
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_onehotCoding
train_y = np.array(list(x_train['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding)).t
test_y = np.array(list(x_test['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(x_cv['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_va
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_varia
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_f

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_re
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_response
```

In [111]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehot
print("(number of data points * number of features) in test data = ", test_x_onehotCo
print("(number of data points * number of features) in cross validation data =", cv_x
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 115580)
(number of data points * number of features) in test data =  (665, 115580)
(number of data points * number of features) in cross validation data = (532, 115580)
```

In [112]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respon
print("(number of data points * number of features) in test data = ", test_x_response
print("(number of data points * number of features) in cross validation data =", cv_x
```

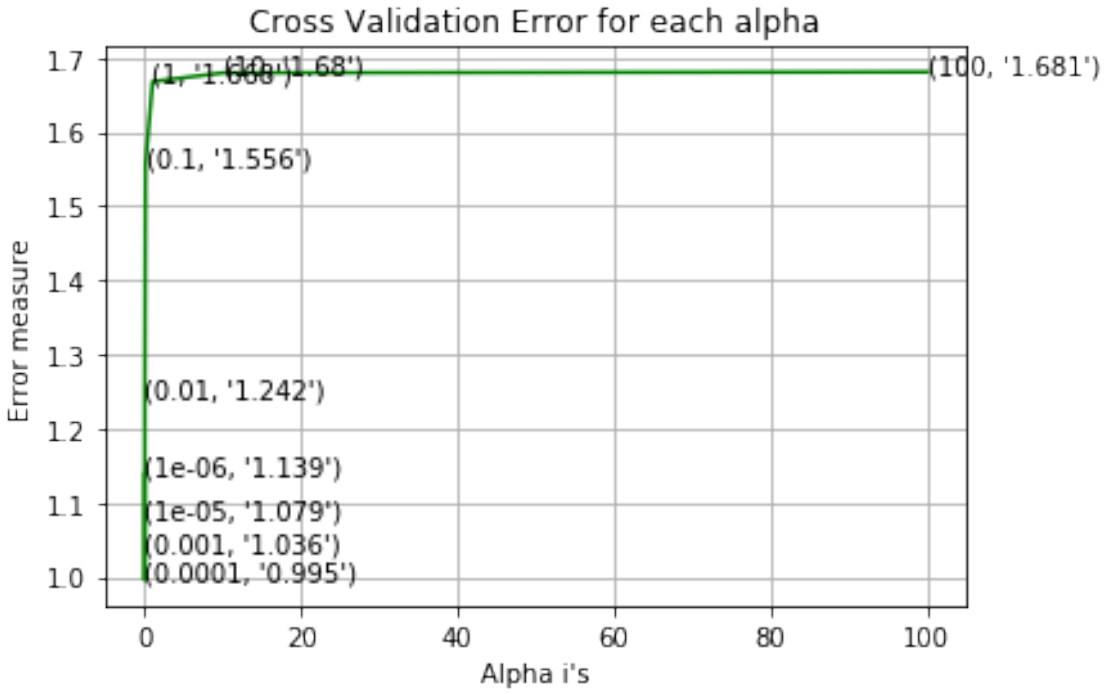
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

In [113]:
```
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', r
                clf.fit(train_x_onehotCoding, train_y)
                sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                sig_clf.fit(train_x_onehotCoding, train_y)
                sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
                cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
                # to avoid rounding error while multiplying probabilites we use log-probability
                print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


            best_alpha = np.argmin(cv_log_error_array)
            clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
            clf.fit(train_x_onehotCoding, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x_onehotCoding, train_y)

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

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

```
for alpha = 1e-06
Log Loss : 1.13862052653
for alpha = 1e-05
```

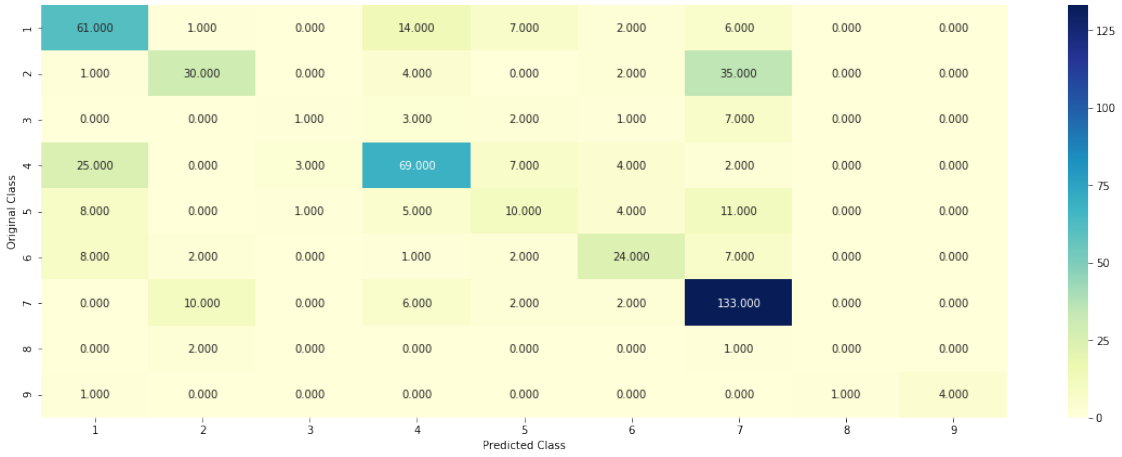
```
Log Loss : 1.07871574942
for alpha = 0.0001
Log Loss : 0.995245278788
for alpha = 0.001
Log Loss : 1.03596950122
for alpha = 0.01
Log Loss : 1.24156544042
for alpha = 0.1
Log Loss : 1.55579759866
for alpha = 1
Log Loss : 1.66777907092
for alpha = 10
Log Loss : 1.67950269795
for alpha = 100
Log Loss : 1.6807669731
```




```
For values of best alpha =  0.0001 The train log loss is: 0.446241135134
For values of best alpha =  0.0001 The cross validation log loss is: 0.995245278788
For values of best alpha =  0.0001 The test log loss is: 0.985257215859
```

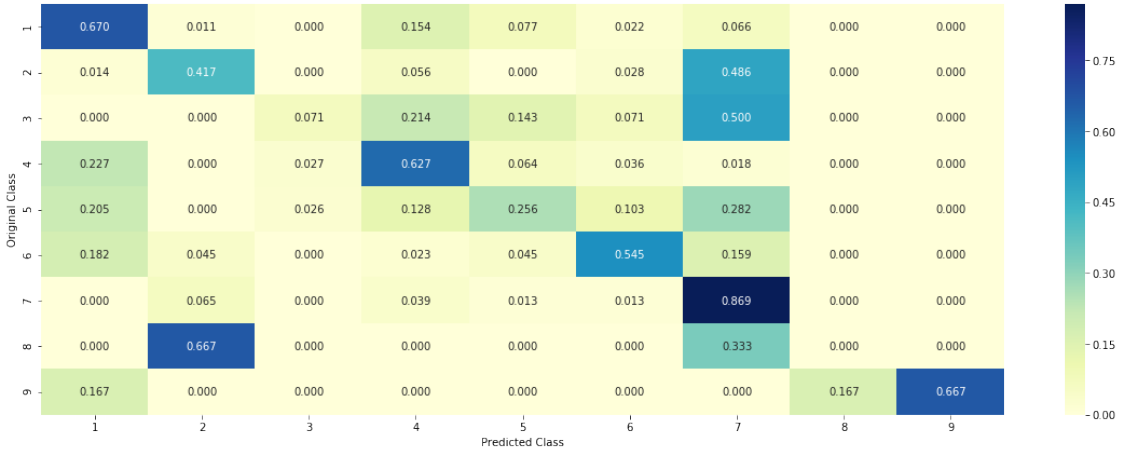
```
In [114]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
          predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, c
```

```
Log loss : 0.995245278788
Number of mis-classified points : 0.37593984962406013

-------------------- Confusion matrix --------------------
```




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


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



After some feature engineering we manage to decrease the log loss below < 1. We can adopt more feature enginnering methods and reduce the log loss furhermore.

0.0.2 Lets summarize above models before proceeding with the feature engineering approach.

```
In [36]: from prettytable import PrettyTable
         ptable = PrettyTable()

         ptable.field_names=["Model Name","Train","CV","Test","% Misclassified Points"]
         ptable.add_row(["Naive Bayes","0.49","1.20","1.21","41"])
         ptable.add_row(["KNN","0.63","1.04","1.06","35"])
         ptable.add_row(["Logistic Regression With Class balancing","0.42","0.98","1.02","32"]
         ptable.add_row(["Logistic Regression Without Class balancing","0.41","1.00","1.05","33
         ptable.add_row(["Linear SVM","0.57","1.01","1.08","32"])
         ptable.add_row(["Random Forest Classifier With One hot Encoding","0.87","1.24","1.24"
         ptable.add_row(["Random Forest Classifier With Response Coding","0.06","1.38","1.38",
         ptable.add_row(["Stack Models:LR+NB+SVM","1.02","1.35","1.42","36"])
         ptable.add_row(["Maximum Voting classifier","0.94","1.11","1.16","36"])
         ptable.add_row(["Logistic Regression with unigram and bigram","0.87","1.23","1.19","41
         ptable.add_row(["Logistic Regression with Feature Engineering","0.44","0.99","0.98","3
         print(ptable)
```

```
+-------------------------------------------------+-------+------+------+----------------------
|                   Model Name                    | Train |  CV  | Test | % Misclassified Points
+-------------------------------------------------+-------+------+------+----------------------
|                  Naive Bayes                    |  0.49 | 1.20 | 1.21 |           41
|                      KNN                        |  0.63 | 1.04 | 1.06 |           35
|    Logistic Regression With Class balancing     |  0.42 | 0.98 | 1.02 |           32
|  Logistic Regression Without Class balancing    |  0.41 | 1.00 | 1.05 |           33
|                  Linear SVM                     |  0.57 | 1.01 | 1.08 |           32
| Random Forest Classifier With One hot Encoding  |  0.87 | 1.24 | 1.24 |           45
| Random Forest Classifier With Response Coding   |  0.06 | 1.38 | 1.38 |           56
```

Stack Models:LR+NB+SVM	1.02	1.35	1.42	36
Maximum Voting classifier	0.94	1.11	1.16	36
Logistic Regression with unigram and bigram	0.87	1.23	1.19	41
Logistic Regression with Feature Engineering	0.44	0.99	0.98	37

From above table we can say that 'Logistic Regression With Class balancing' works better than other model.

0.1 Steps Followed

1. Store the data from train_variants and text_variants in the variables.
2. After that we perform de-duplication and then preprocess and clean it to remove unwanted/corrupted data.
3. Then perform univariate data analysis on all column to understand more about it i.e. which feature are more important than the other.
4. Then we build various machine learning models on top of the processed data.
5. After that we perform CountVectorizer with both unigram and bigram.
6. At the last we performed feature engineering(like mergine the column) to reduce the log-loss<1.