# CancerDiagnosis

August 4, 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
        from nltk.corpus import stopwords
        from sklearn.decomposition import TruncatedSVD
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.manifold import TSNE
        import seaborn as sns
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics.classification import accuracy_score, log_loss
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.linear_model import SGDClassifier
        from imblearn.over_sampling import SMOTE
        from collections import Counter
        from scipy.sparse import hstack
        from sklearn.multiclass import OneVsRestClassifier
        from sklearn.svm import SVC
        # from sklearn.cross_validation import StratifiedKFold
        from sklearn.model_selection import StratifiedKFold
        from collections import Counter, defaultdict
        from sklearn.calibration import CalibratedClassifierCV
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.naive_bayes import GaussianNB
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import GridSearchCV
        import math
        from sklearn.metrics import normalized_mutual_info_score
        from sklearn.ensemble import RandomForestClassifier
        warnings.filterwarnings("ignore")

        from mlxtend.classifier import StackingClassifier

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

### 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

```
In [2]: data = pd.read_csv('training/training_variants')
        print('Number of data points : ', data.shape[0])
        print('Number of features : ', data.shape[1])
        print('Features : ', data.columns.values)
        data.head()
```

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

```
Out[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("training/training_text",sep="\|\|",engine="python",names=["ID"
        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
        stop_words = set(stopwords.words('english'))


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

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

                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")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
```

```
Time took for preprocessing the text : 142.199534 seconds
```

```
In [6]: #merging both gene_variations and text data based on ID
        result = pd.merge(data, 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 dependent kinases cdks regulate variety...
        1  abstract background non small cell lung cancer...
        2  abstract background non small cell lung cancer...
        3  recent evidence demonstrated acquired uniparen...
        4  oncogenic mutations monomeric casitas b lineag...
```

```
In [7]: result[result.isnull().any(axis=1)]
```

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

```
In [8]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [9]: result[result['ID']==1109]
```

```
Out[9]:         ID   Gene Variation  Class          TEXT
        1109  1109  FANCA    S1088F      1  FANCA S1088F
```

### 3.1.4. Test, Train and Cross Validation Split

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

```
In [10]: y_true = result['Class'].values
         result.Gene      = result.Gene.str.replace('\s+', '_')
         result.Variation = result.Variation.str.replace('\s+', '_')

         # split the data into test and train by maintaining same distribution of output varai
         X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true,
         # split the train data into train and cross validation by maintaining same distributi
         train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train,
```

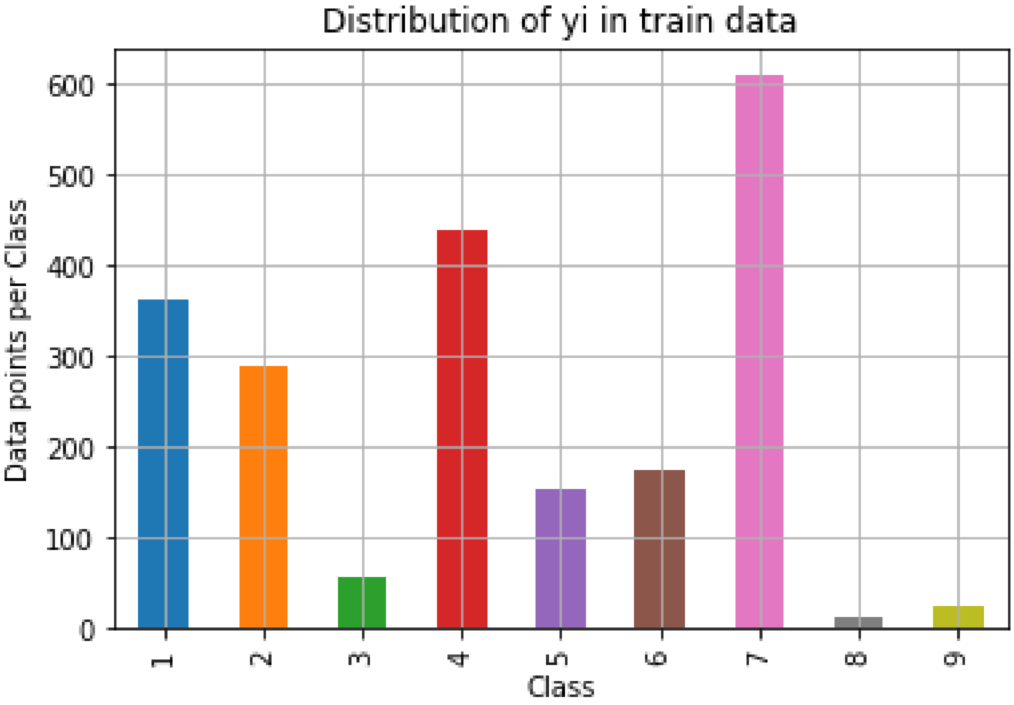
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

```
In [11]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

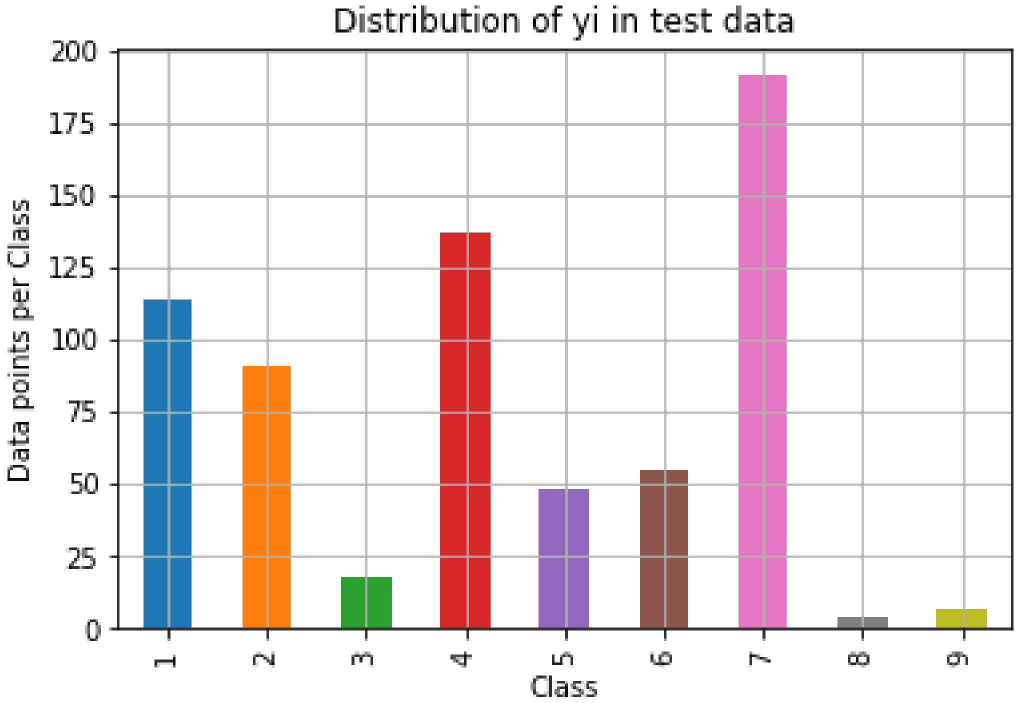
### 3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [12]: # it returns a dict, keys as class labels and values as the number of data points in
         train_class_distribution = train_df['Class'].value_counts().sortlevel()
         test_class_distribution = test_df['Class'].value_counts().sortlevel()
         cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

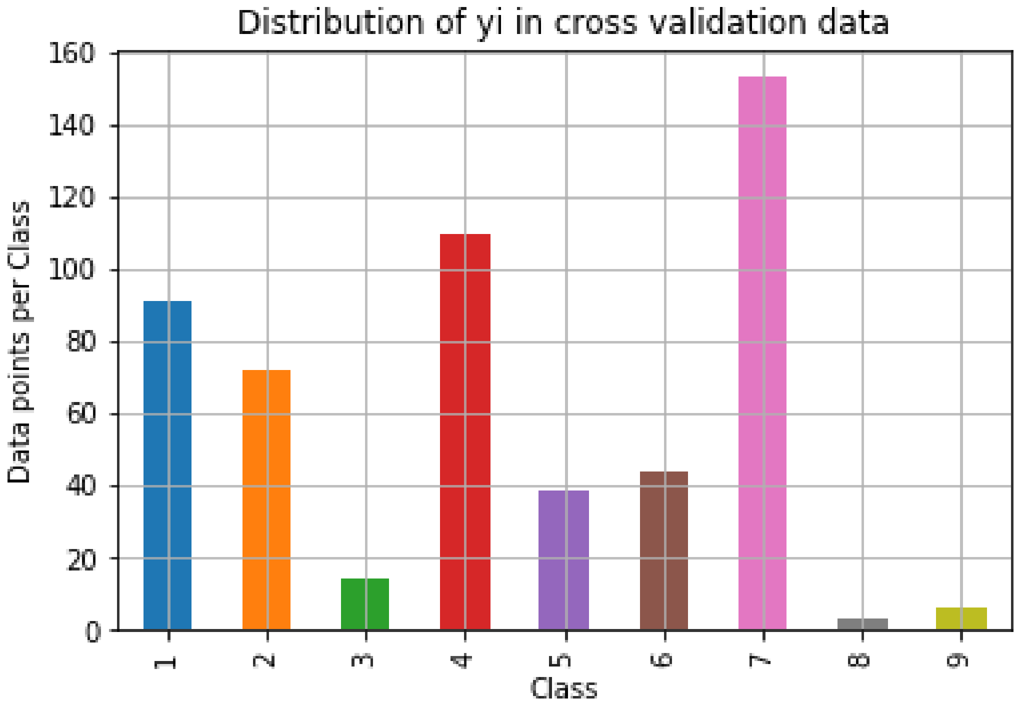


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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

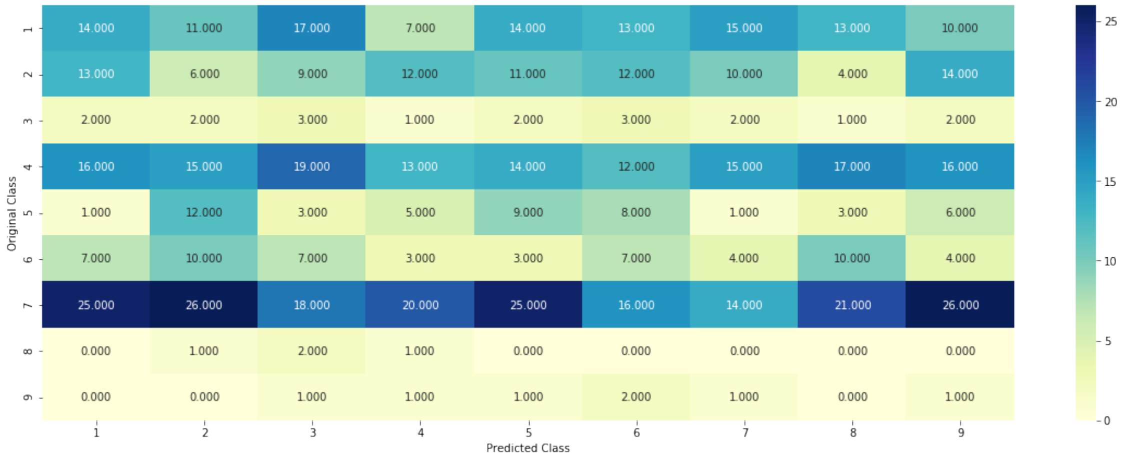
In a ’Random’ Model, we generate the NINE class probabilites randomly such that they sum to 1.

```
In [13]: # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
             # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

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

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

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

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

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

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

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

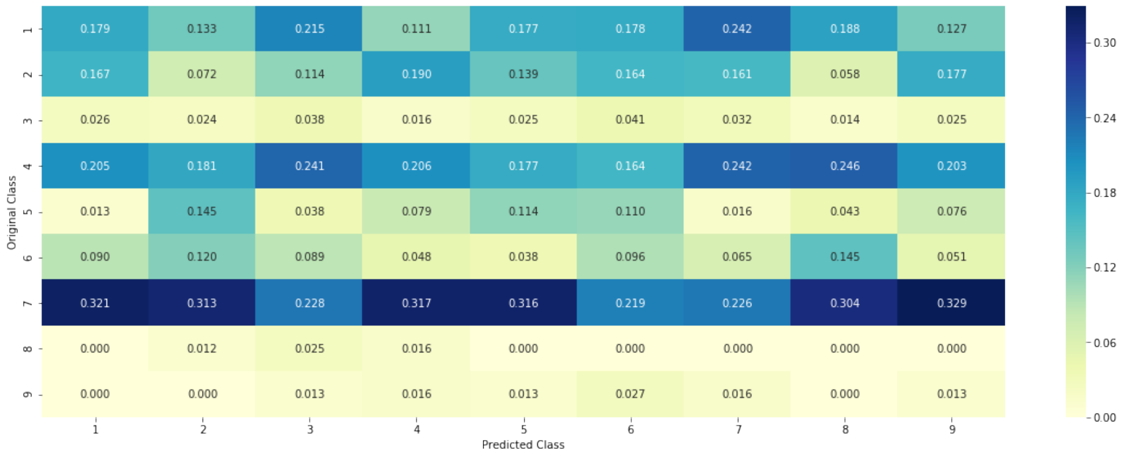
```
In [14]: # we need to generate 9 numbers and the sum of numbers should be 1
         # one solution is to genarate 9 numbers and divide each of the numbers by their sum
         # ref: https://stackoverflow.com/a/18662466/4084039
```

```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


# 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)
```

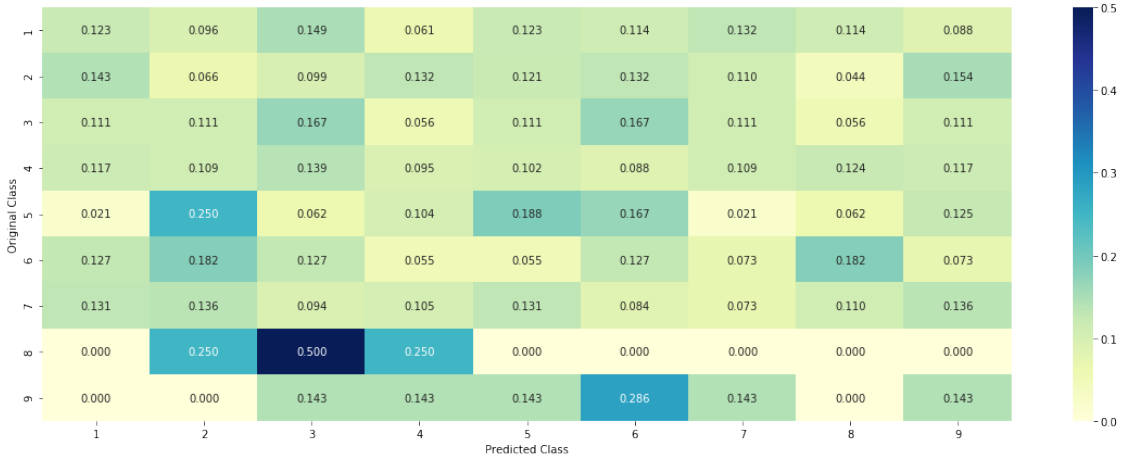
```
Log loss on Cross Validation Data using Random Model 2.510853375842832
Log loss on Test Data using Random Model 2.4829695422743243
-------------------- Confusion matrix --------------------
```



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



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



### 3.3 Univariate Analysis

```
In [15]: # code for response coding with Laplace smoothing.
         # alpha : used for laplace smoothing
         # feature: ['gene', 'variation']
         # df: ['train_df', 'test_df', 'cv_df']
         # algorithm
         # ----------
         # Consider all unique values and the number of occurances of given 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 = train_df[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 t
    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 = train_df.loc[(train_df['Class']==k) & (train_df[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, 0.03787878787878788, 0.068181818181818177,
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818181818181
    #      '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 = train_df[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?

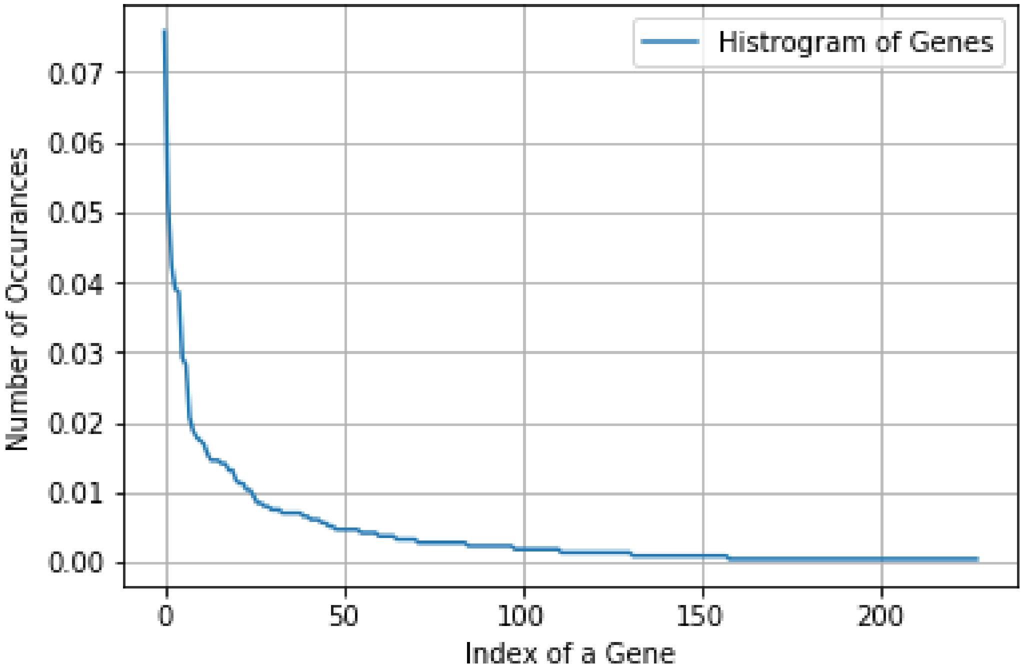
```
In [16]: unique_genes = train_df['Gene'].value_counts()
         print('Number of Unique Genes :', unique_genes.shape[0])
         # the top 10 genes that occured most
         print(unique_genes.head(10))
```

```
Number of Unique Genes : 228
BRCA1     161
TP53      109
BRCA2      90
PTEN       83
EGFR       82
BRAF       62
KIT        60
ERBB2      44
ALK        40
PDGFRA     38
Name: Gene, dtype: int64
```

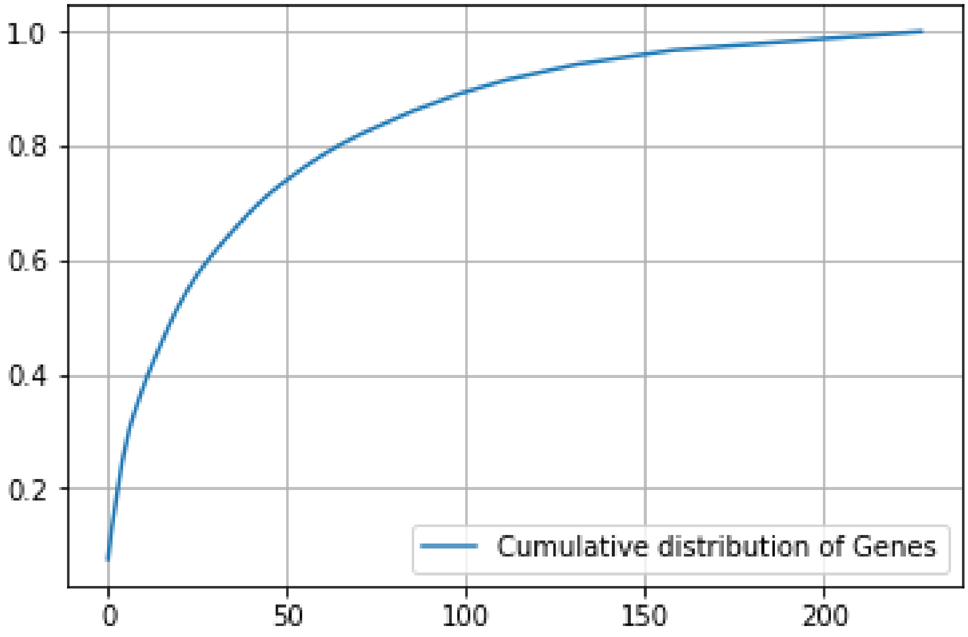
```
In [17]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the 
```

```
Ans: There are 228 different categories of genes in the train data, and they are distibuted as
```

```
In [18]: 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 [19]: c = np.cumsum(h)
         plt.plot(c,label='Cumulative distribution of Genes')
         plt.grid()
         plt.legend()
         plt.show()
```



Q3. How to featurize this Gene feature ?

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

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

```
In [21]: print("train_gene_feature_responseCoding is converted feature using respone coding me
```

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

```
In [22]: # one-hot encoding of Gene feature.
         gene_vectorizer = CountVectorizer()
         train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
         test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
         cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

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

```
Out[23]: 2906      NF2
         2966      KIT
         1675     FLT3
         2359    STK11
         2608    BRCA1
         Name: Gene, dtype: object
```

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

```
Out[24]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
'akt3',
'alk',
'apc',
'ar',
'araf',
'arid1a',
'arid2',
'arid5b',
'asxl2',
'atm',
'atr',
'atrx',
'aurka',
'axin1',
'axl',
'b2m',
'bap1',
'bard1',
'bcl10',
'bcl2',
'bcor',
'braf',
'brca1',
'brca2',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd2',
'ccnd3',
'cdh1',
'cdk12',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3b',
'egfr',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
    'ikbke',
    'ikzf1',
    'jak1',
    'jak2',
    'kdm5a',
    'kdm5c',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'klf4',
    'kmt2a',
    'kmt2b',
    'kmt2c',
    'kmt2d',
    'knstrn',
    'kras',
    'lats1',
    'lats2',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mapk1',
    'mdm2',
    'mdm4',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mga',
    'mlh1',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myd88',
    'nf1',
    'nf2',
    'nfe2l2',
    'nfkbia',
    'nkx2',
    'notch1',
    'notch2',
    'npm1',
    'nras',
    'nsd1',
    'ntrk1',
    'ntrk2',
    'ntrk3',
    'nup93',
    'pak1',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
    'pik3r3',
    'pim1',
    'pms1',
    'pms2',
    'pole',
    'ppm1d',
    'ppp2r1a',
    'ppp6c',
    'prdm1',
    'ptch1',
    'pten',
    'ptpn11',
    'ptprd',
    'ptprt',
    'rab35',
    'rac1',
    'rad21',
    'rad50',
    'rad51c',
    'rad51d',
    'raf1',
    'rara',
    'rasa1',
    'rb1',
    'rbm10',
    'ret',
    'rheb',
    'rhoa',
    'rit1',
    'rnf43',
    'ros1',
    'rras2',
    'runx1',
    'rxra',
    'rybp',
    'sdhb',
         'setd2',
         'sf3b1',
         'shoc2',
         'smad2',
         'smad3',
         'smad4',
         'smarca4',
         'smo',
         'sos1',
         'sox9',
         'spop',
         'src',
         'srsf2',
         'stag2',
         'stat3',
         'stk11',
         'tert',
         'tet1',
         'tet2',
         'tgfbr1',
         'tgfbr2',
         'tmprss2',
         'tp53',
         'tp53bp1',
         'tsc1',
         'tsc2',
         'u2af1',
         'vegfa',
         'vhl',
         'xrcc2',
         'yap1']
```

```
In [25]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding me
```

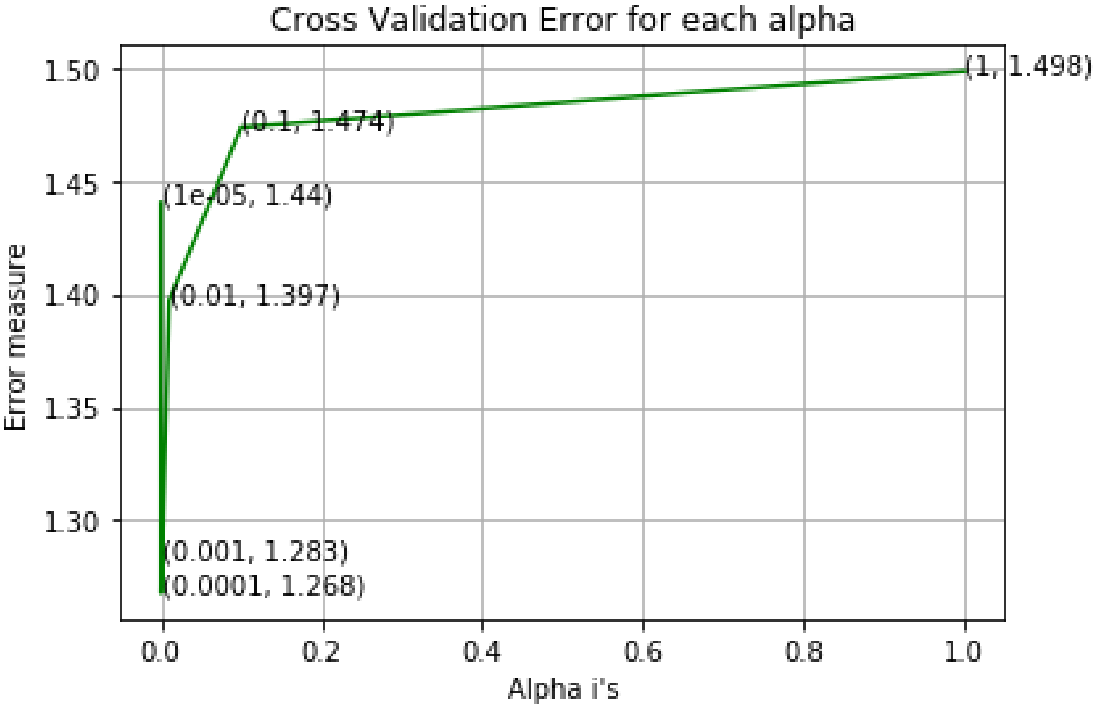
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape o
```

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-1
             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=4
         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_l
         predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
         predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.440470611887944
For values of alpha =  0.0001 The log loss is: 1.2680300486887892
For values of alpha =  0.001 The log loss is: 1.2830792520479086
For values of alpha =  0.01 The log loss is: 1.3970477449098202
For values of alpha =  0.1 The log loss is: 1.4736833142498464
For values of alpha =  1 The log loss is: 1.4983476782644036
```



```
For values of best alpha =  0.0001 The train log loss is: 1.02833261997304
For values of best alpha =  0.0001 The cross validation log loss is: 1.2680300486887892
For values of best alpha =  0.0001 The test log loss is: 1.2286393900363202
```

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 [27]: print("Q6. How many data points in Test and CV datasets are covered by the ", unique_

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  228  genes in train datase
Ans
1. In test data 639 out of 665 : 96.09022556390977
2. In cross validation data 507 out of  532 : 95.30075187969925
```

### 3.2.2 Univariate Analysis on Variation Feature

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

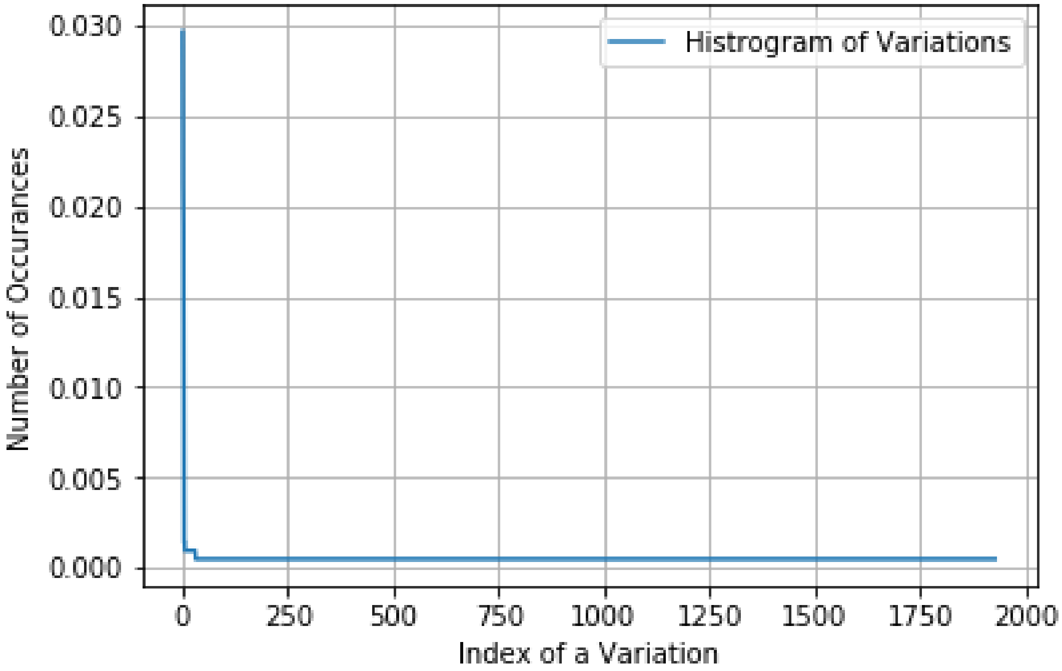
```
In [28]: unique_variations = train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         # the top 10 variations that occured most
         print(unique_variations.head(10))
```

```
Number of Unique Variations : 1929
Truncating_Mutations    63
Deletion                45
Amplification           40
Fusions                 22
Q61H                     3
T58I                     3
A146T                    2
Q209L                    2
G13V                     2
Overexpression           2
Name: Variation, dtype: int64
```

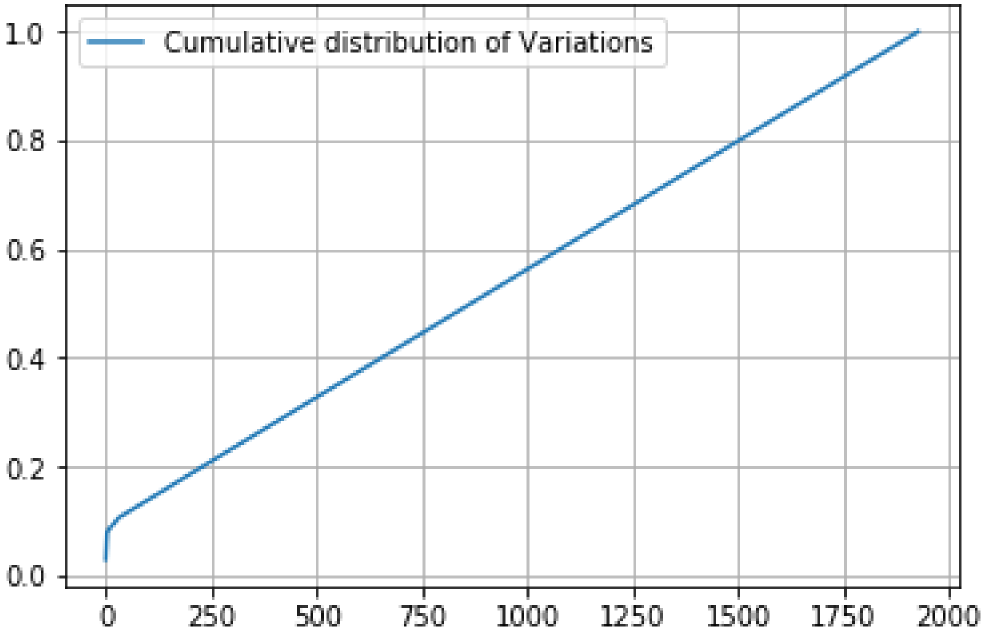
```
In [29]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variatio
```

```
Ans: There are 1929 different categories of variations in the train data, and they are distibu
```

```
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()

[0.02966102 0.05084746 0.06967985 ... 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 [32]: # alpha is used for laplace smoothing
         alpha = 1
         # train gene feature
         train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
         # test gene feature
         test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
         # cross validation gene feature
         cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_c
```

```
In [33]: print("train_variation_feature_responseCoding is a converted feature using the respons

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

```
In [34]: # one-hot encoding of variation feature.
         variation_vectorizer = CountVectorizer()
         train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
         test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
         cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation']]
In [35]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot e
```

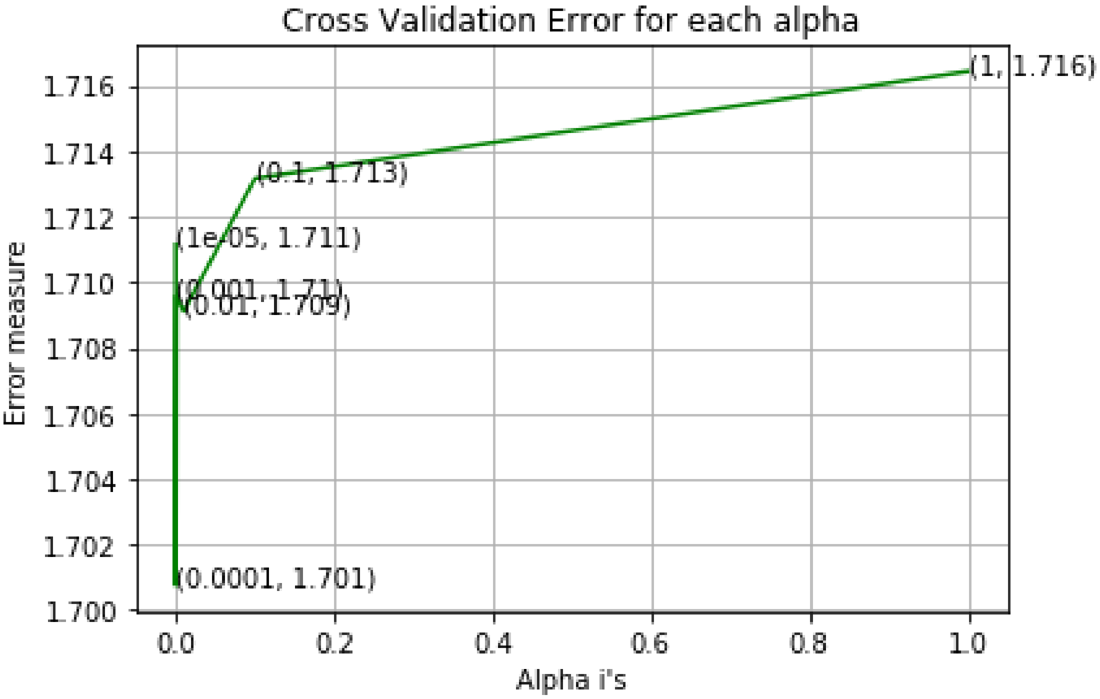
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
```

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: 1.7111557675562805
For values of alpha =  0.0001 The log loss is: 1.7007541678303986
For values of alpha =  0.001 The log loss is: 1.7095955962408393
For values of alpha =  0.01 The log loss is: 1.709111824934141
For values of alpha =  0.1 The log loss is: 1.7131648413540987
For values of alpha =  1 The log loss is: 1.7164268050341223
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7358096065669432
For values of best alpha =  0.0001 The cross validation log loss is: 1.7007541678303986
For values of best alpha =  0.0001 The test log loss is: 1.6944517261713599
```

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], '
         test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].sha
         cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
         print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_cove
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_co
```

```
Q12. How many data points are covered by total  1929  genes in test and cross validation data s
Ans
1. In test data 77 out of 665 : 11.578947368421053
2. In cross validation data 49 out of  532 : 9.210526315789473
```

### 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 [38]: # 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 [39]: 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 [40]: from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.feature_selection import chi2
         from sklearn.feature_selection import SelectKBest


         # 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(train_df['TEXT'])
         # getting all the feature names (words)
         train_text_features= text_vectorizer.get_feature_names()

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

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

         # taking TOP 1000 features
         train_text_feature_onehotCoding = SelectKBest(chi2, k=1000).fit_transform(train_text_

         print("Total number of unique words in train data :", len(train_text_features))
         print('train_text_feature_onehotCoding shape ==>',train_text_feature_onehotCoding.sha
```

```
Total number of unique words in train data : 1000
train_text_feature_onehotCoding shape ==> (2124, 1000)
```

```
In [41]: train_text_features[:10], text_fea_dict.get('000')

Out[41]: (['00', '000', '01', '05', '10', '100', '11', '12', '13', '14'],
          14.783310243005301)
```

```
In [42]: dict_list = []
         # dict_list =[] contains 9 dictoinaries each corresponds to a class
         for i in range(1,10):
             cls_text = train_df[train_df['Class']==i]
             # build a word dict based on the words in that class
             dict_list.append(extract_dictionary_paddle(cls_text))
             # append it to dict_list

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

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

```
In [43]: # dict_list[1]
         # confuse_array[0],confuse_array[1],confuse_array[2],confuse_array[3],confuse_array[4
         len(confuse_array)
```

```
Out[43]: 1000
```

```
In [44]: #response coding of text features
         train_text_feature_responseCoding  = get_text_responsecoding(train_df)
         test_text_feature_responseCoding  = get_text_responsecoding(test_df)
         cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

```
In [45]: # 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 [46]: # don't forget to normalize every feature
         train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```
In [47]: #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()))
```

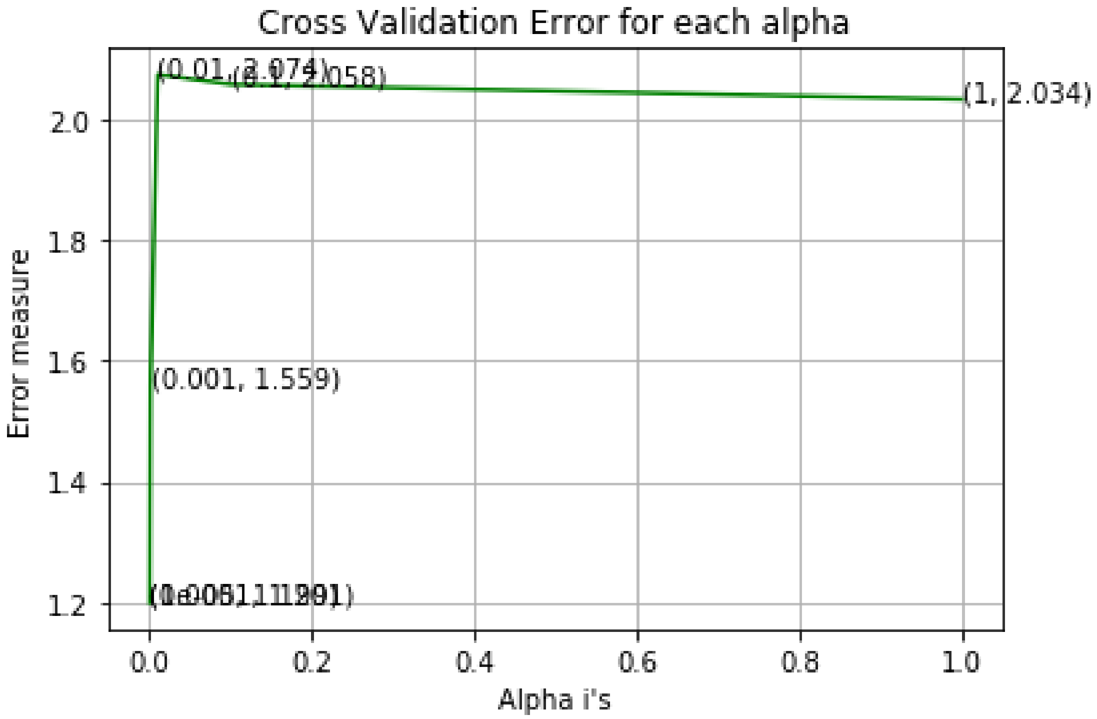
```
In [48]: # Number of words for a given frequency.
         print(Counter(sorted_text_occur))
```

```
Counter({251.44141104438117: 1, 175.341660939697: 1, 136.20164752478405: 1, 130.17587367212175
```

In [49]:
```
# 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-1
    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=4
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: 1.1990172325576594
For values of alpha =  0.0001 The log loss is: 1.2011063515349214
For values of alpha =  0.001 The log loss is: 1.55856814414325
For values of alpha =  0.01 The log loss is: 2.074453907622877
For values of alpha =  0.1 The log loss is: 2.0583897813570475
For values of alpha =  1 The log loss is: 2.03365752438238
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7781768854069671
For values of best alpha =  1e-05 The cross validation log loss is: 1.1990172325576594
For values of best alpha =  1e-05 The test log loss is: 1.1743373045965237
```

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

```
In [51]: def get_intersec_text(df):
             df_text_vec = CountVectorizer(min_df=3)
```

```
        df_text_fea = df_text_vec.fit_transform(df['TEXT'])
        df_text_features = df_text_vec.get_feature_names()

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

```
In [52]: len1,len2 = get_intersec_text(test_df)
         print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
         len1,len2 = get_intersec_text(cv_df)
         print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train
```

```
3.448 % of word of test data appeared in train data
3.79 % of word of Cross Validation appeared in train data
```

## 4. Machine Learning Models

```
In [52]: #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 [53]: 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 [54]: # this function will be used just for naive bayes
         # for the given indices, we will print the name of the features
         # and we will check whether the feature present in the test point text or not
         def get_impfeature_names(indices, text, gene, var, no_features):
             gene_count_vec = CountVectorizer()
             var_count_vec = CountVectorizer()
```

```
    text_count_vec = TfidfVectorizer(min_df=3, max_features=1000)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(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 [55]: # 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_
         train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehot
         train_y = np.array(list(train_df['Class']))

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

         cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).
         cv_y = np.array(list(cv_df['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 [56]: 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, 3195)
(number of data points * number of features) in test data =  (665, 3195)
(number of data points * number of features) in cross validation data = (532, 3195)
```

```
In [57]: 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_response
         print("(number of data points * number of features) in cross validation data =", cv_x
```

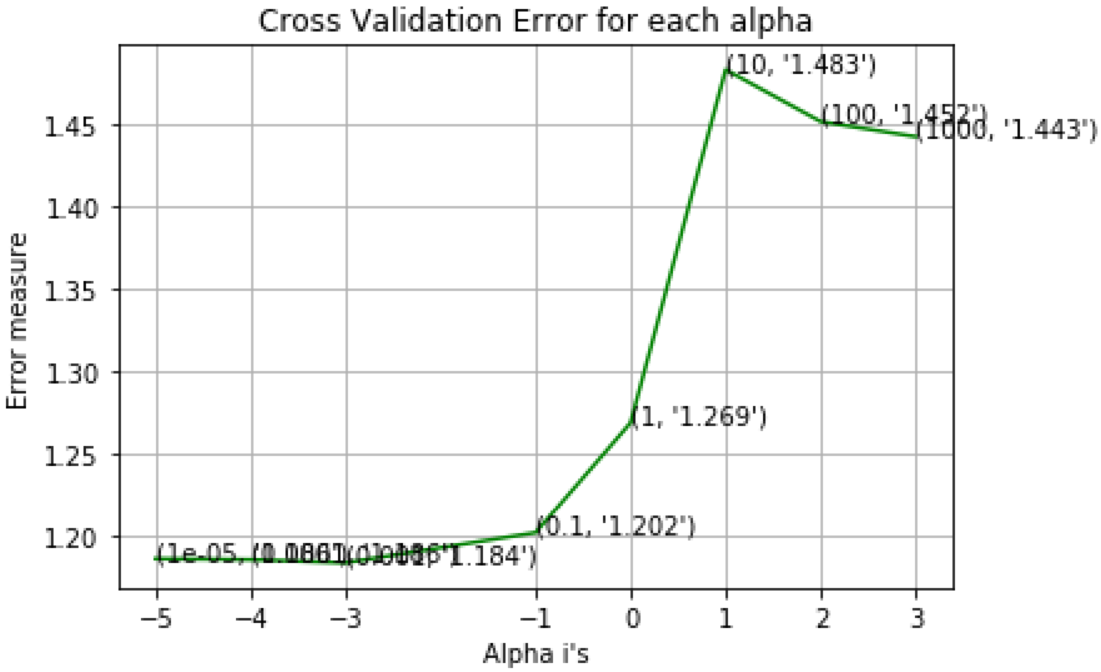
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

```
In [59]: # 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))

         print('=='*25)
         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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 1e-05
Log Loss : 1.18618885722157
for alpha = 0.0001
Log Loss : 1.1856737416405576
for alpha = 0.001
Log Loss : 1.1837169890064192
for alpha = 0.1
Log Loss : 1.2020149288803315
for alpha = 1
Log Loss : 1.26890842893281
for alpha = 10
Log Loss : 1.4834280539786144
for alpha = 100
Log Loss : 1.4519007480265433
for alpha = 1000
Log Loss : 1.44281256571097
==================================================
```

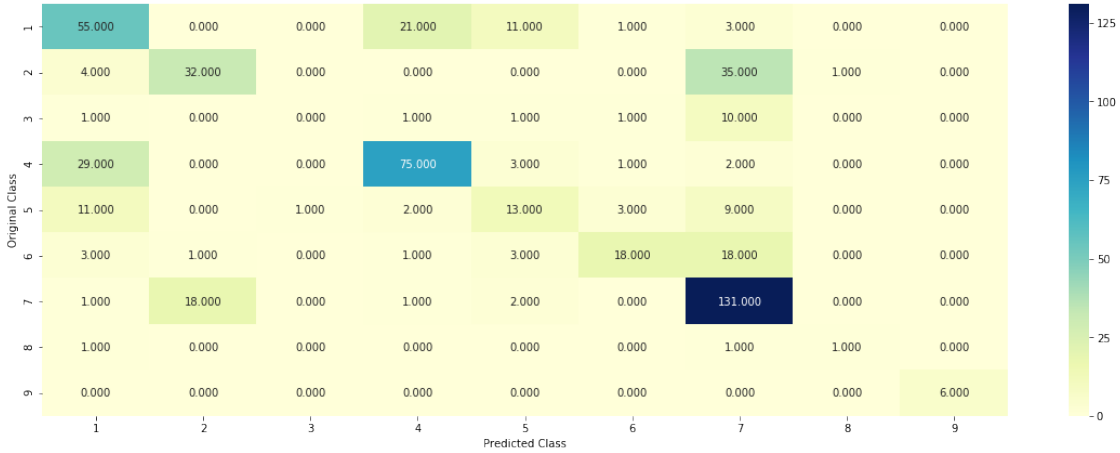


```
For values of best alpha =  0.001 The train log loss is: 0.5234270068824562
For values of best alpha =  0.001 The cross validation log loss is: 1.1837169890064192
For values of best alpha =  0.001 The test log loss is: 1.1776485022515406
```

#### 4.1.1.2. Testing the model with best hyper paramters

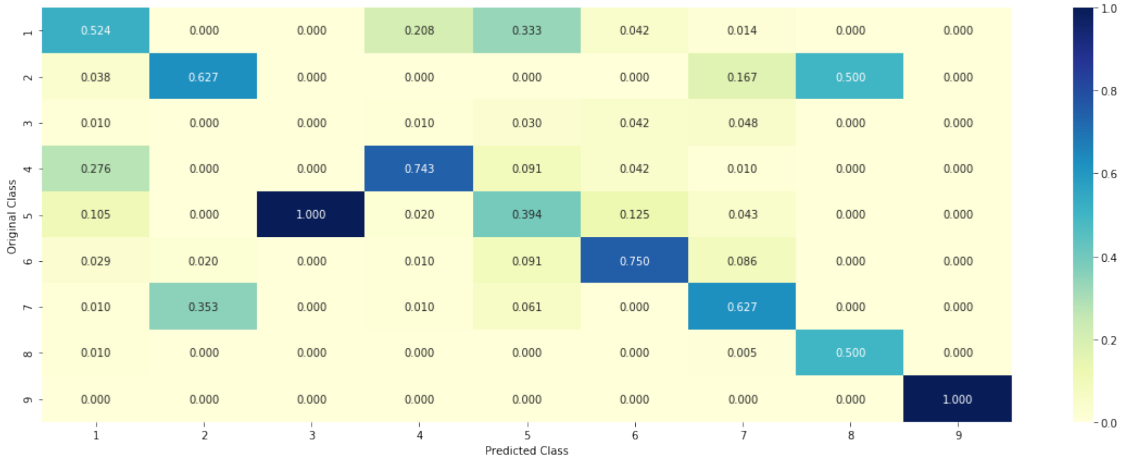
```
In [60]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable
         # -------------------------
         # default paramters
         # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

         # 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 : 1.1837169890064192
Number of missclassified point : 0.37781954887218044
-------------------- Confusion matrix --------------------
```

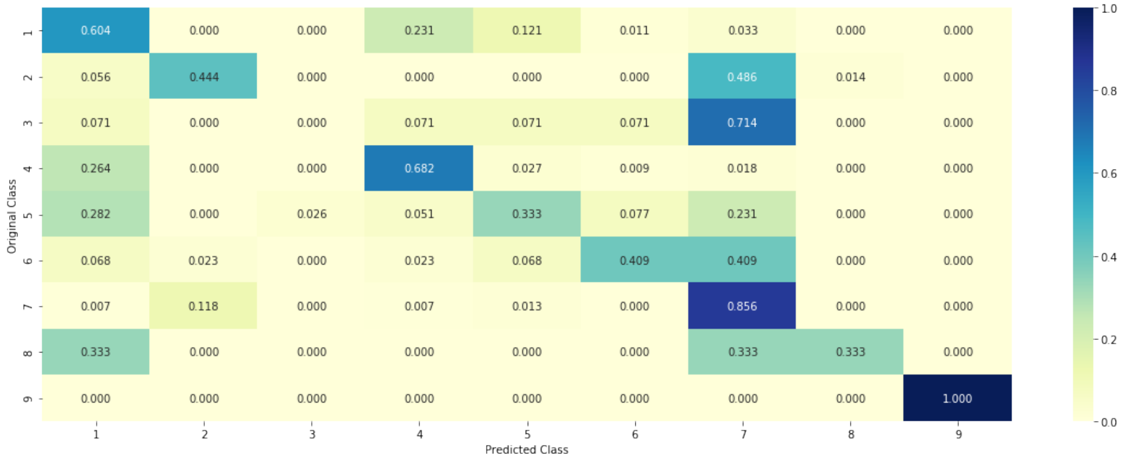


| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 55.000 | 0.000 | 0.000 | 21.000 | 11.000 | 1.000 | 3.000 | 0.000 | 0.000 |
| 2 | 4.000 | 32.000 | 0.000 | 0.000 | 0.000 | 0.000 | 35.000 | 1.000 | 0.000 |
| 3 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 10.000 | 0.000 | 0.000 |
| 4 | 29.000 | 0.000 | 0.000 | 75.000 | 3.000 | 1.000 | 2.000 | 0.000 | 0.000 |
| 5 | 11.000 | 0.000 | 1.000 | 2.000 | 13.000 | 3.000 | 9.000 | 0.000 | 0.000 |
| 6 | 3.000 | 1.000 | 0.000 | 1.000 | 3.000 | 18.000 | 18.000 | 0.000 | 0.000 |
| 7 | 1.000 | 18.000 | 0.000 | 1.000 | 2.000 | 0.000 | 131.000 | 0.000 | 0.000 |
| 8 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 |
| 9 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 6.000 |

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



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



### 4.1.1.3. Feature Importance, Correctly classified point

```
In [61]: test_point_index = 1
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
Predicted Class : 4
Predicted Class Probabilities: [[0.2156 0.0432 0.0125 0.5692 0.0342 0.0321 0.0863 0.0029 0.004
Actual Class : 4
--------------------------------------------------
10 Text feature [activity] present in test data point [True]
11 Text feature [protein] present in test data point [True]
15 Text feature [function] present in test data point [True]
16 Text feature [proteins] present in test data point [True]
17 Text feature [experiments] present in test data point [True]
18 Text feature [acid] present in test data point [True]
19 Text feature [results] present in test data point [True]
20 Text feature [missense] present in test data point [True]
21 Text feature [amino] present in test data point [True]
24 Text feature [also] present in test data point [True]
25 Text feature [whereas] present in test data point [True]
26 Text feature [shown] present in test data point [True]
27 Text feature [type] present in test data point [True]
28 Text feature [whether] present in test data point [True]
29 Text feature [functional] present in test data point [True]
30 Text feature [mutations] present in test data point [True]
31 Text feature [mammalian] present in test data point [True]
33 Text feature [two] present in test data point [True]
34 Text feature [wild] present in test data point [True]
35 Text feature [indicate] present in test data point [True]
36 Text feature [reduced] present in test data point [True]
37 Text feature [previously] present in test data point [True]
38 Text feature [determined] present in test data point [True]
39 Text feature [important] present in test data point [True]
40 Text feature [described] present in test data point [True]
41 Text feature [may] present in test data point [True]
42 Text feature [30] present in test data point [True]
44 Text feature [loss] present in test data point [True]
45 Text feature [either] present in test data point [True]
46 Text feature [therefore] present in test data point [True]
48 Text feature [although] present in test data point [True]
49 Text feature [three] present in test data point [True]
50 Text feature [analysis] present in test data point [True]
51 Text feature [associated] present in test data point [True]
52 Text feature [containing] present in test data point [True]
54 Text feature [show] present in test data point [True]
55 Text feature [suppressor] present in test data point [True]
58 Text feature [determine] present in test data point [True]
59 Text feature [indicated] present in test data point [True]
61 Text feature [affect] present in test data point [True]
62 Text feature [tagged] present in test data point [True]
65 Text feature [one] present in test data point [True]
66 Text feature [suggesting] present in test data point [True]
67 Text feature [discussion] present in test data point [True]
```

```
68 Text feature [found] present in test data point [True]
70 Text feature [mutation] present in test data point [True]
72 Text feature [addition] present in test data point [True]
73 Text feature [however] present in test data point [True]
74 Text feature [vitro] present in test data point [True]
75 Text feature [lower] present in test data point [True]
76 Text feature [similar] present in test data point [True]
77 Text feature [using] present in test data point [True]
78 Text feature [used] present in test data point [True]
79 Text feature [analyzed] present in test data point [True]
80 Text feature [ability] present in test data point [True]
81 Text feature [generated] present in test data point [True]
82 Text feature [cells] present in test data point [True]
83 Text feature [suggest] present in test data point [True]
84 Text feature [indicates] present in test data point [True]
85 Text feature [resulting] present in test data point [True]
86 Text feature [high] present in test data point [True]
87 Text feature [10] present in test data point [True]
88 Text feature [control] present in test data point [True]
89 Text feature [contribute] present in test data point [True]
90 Text feature [buffer] present in test data point [True]
91 Text feature [effects] present in test data point [True]
92 Text feature [acids] present in test data point [True]
94 Text feature [transfection] present in test data point [True]
95 Text feature [could] present in test data point [True]
96 Text feature [bind] present in test data point [True]
97 Text feature [mutant] present in test data point [True]
98 Text feature [stability] present in test data point [True]
99 Text feature [reported] present in test data point [True]
Out of the top  100  features  73 are present in query point
```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

```
In [62]: test_point_index = 100
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0579 0.0465 0.0113 0.0877 0.031  0.0293 0.7295 0.0029 0.003
Actual Class : 7
--------------------------------------------------
```

```
18 Text feature [activation] present in test data point [True]
19 Text feature [activated] present in test data point [True]
20 Text feature [kinase] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
23 Text feature [inhibitor] present in test data point [True]
24 Text feature [cells] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [presence] present in test data point [True]
27 Text feature [factor] present in test data point [True]
28 Text feature [contrast] present in test data point [True]
29 Text feature [signaling] present in test data point [True]
30 Text feature [treatment] present in test data point [True]
31 Text feature [treated] present in test data point [True]
32 Text feature [independent] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [constitutive] present in test data point [True]
37 Text feature [growth] present in test data point [True]
38 Text feature [phosphorylation] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [shown] present in test data point [True]
43 Text feature [well] present in test data point [True]
44 Text feature [addition] present in test data point [True]
45 Text feature [higher] present in test data point [True]
46 Text feature [similar] present in test data point [True]
48 Text feature [compared] present in test data point [True]
49 Text feature [recently] present in test data point [True]
50 Text feature [mutations] present in test data point [True]
51 Text feature [cell] present in test data point [True]
52 Text feature [tyrosine] present in test data point [True]
53 Text feature [showed] present in test data point [True]
55 Text feature [found] present in test data point [True]
56 Text feature [may] present in test data point [True]
58 Text feature [although] present in test data point [True]
62 Text feature [previously] present in test data point [True]
64 Text feature [increased] present in test data point [True]
65 Text feature [suggest] present in test data point [True]
66 Text feature [without] present in test data point [True]
68 Text feature [pathway] present in test data point [True]
70 Text feature [survival] present in test data point [True]
71 Text feature [proliferation] present in test data point [True]
72 Text feature [pathways] present in test data point [True]
73 Text feature [mutation] present in test data point [True]
74 Text feature [20] present in test data point [True]
75 Text feature [15] present in test data point [True]
76 Text feature [enhanced] present in test data point [True]
77 Text feature [absence] present in test data point [True]
80 Text feature [including] present in test data point [True]
81 Text feature [studies] present in test data point [True]
```

```
83 Text feature [results] present in test data point [True]
84 Text feature [interestingly] present in test data point [True]
85 Text feature [mutant] present in test data point [True]
88 Text feature [increase] present in test data point [True]
89 Text feature [fig] present in test data point [True]
90 Text feature [either] present in test data point [True]
92 Text feature [identified] present in test data point [True]
93 Text feature [various] present in test data point [True]
94 Text feature [13] present in test data point [True]
95 Text feature [lines] present in test data point [True]
97 Text feature [described] present in test data point [True]
98 Text feature [examined] present in test data point [True]
Out of the top  100  features  60 are present in query point
```

### 4.2. K Nearest Neighbour Classification

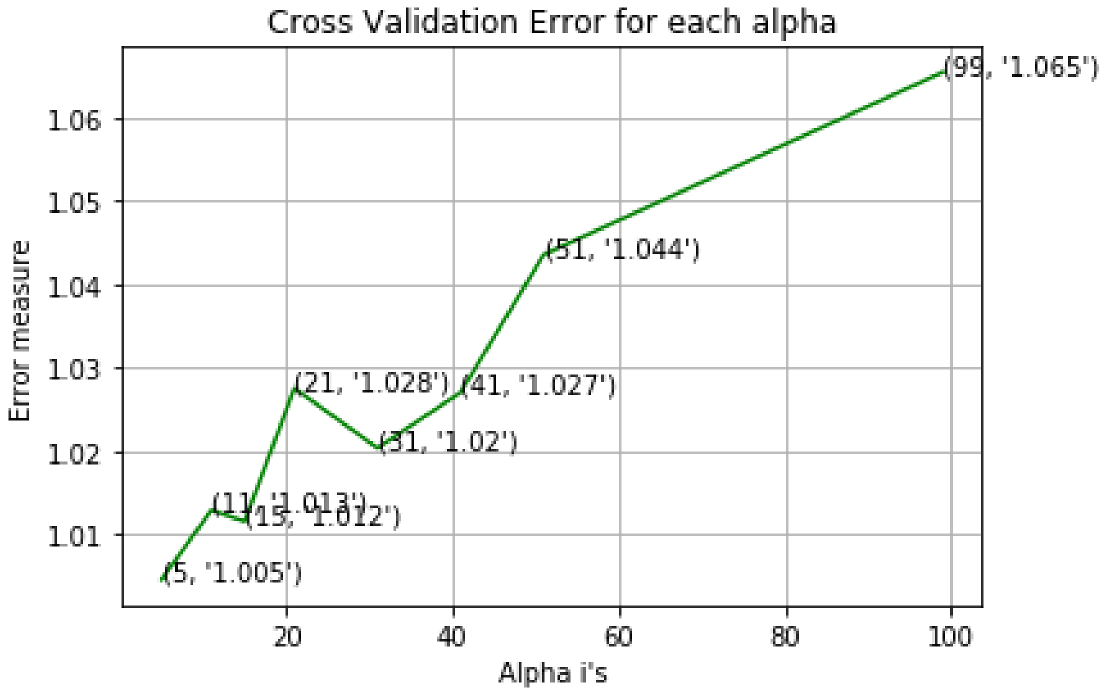
#### 4.2.1. Hyper parameter tuning

```
In [63]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
         # -------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
         # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

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


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

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

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


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

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

```
for alpha = 5
Log Loss : 1.0046408472971262
for alpha = 11
Log Loss : 1.0129559717309329
for alpha = 15
Log Loss : 1.01161728525124
for alpha = 21
Log Loss : 1.0275526018549699
for alpha = 31
Log Loss : 1.020394292823489
for alpha = 41
```

```
Log Loss : 1.0270641321706675
for alpha = 51
Log Loss : 1.0435689046882037
for alpha = 99
Log Loss : 1.0654120600196628
```



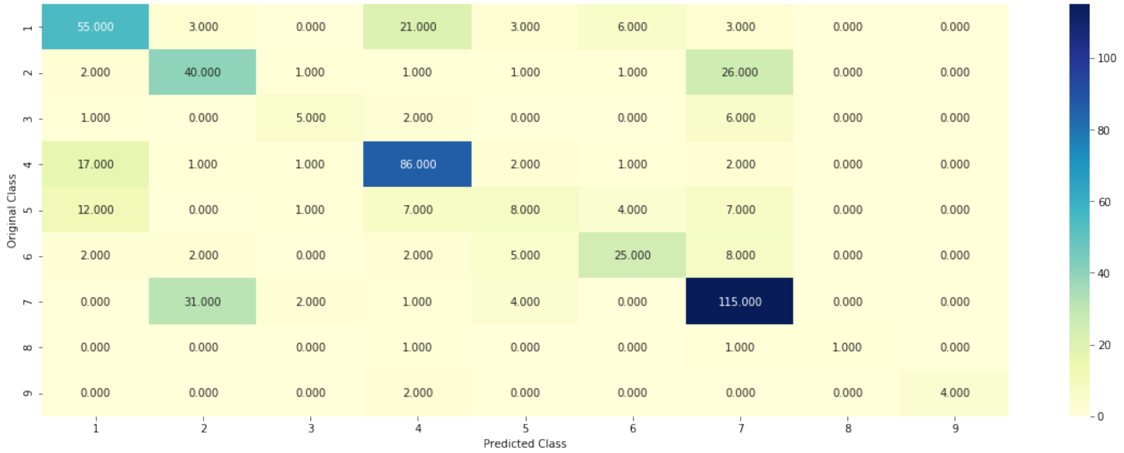
```
For values of best alpha =  5 The train log loss is: 0.507996008911384
For values of best alpha =  5 The cross validation log loss is: 1.0046408472971262
For values of best alpha =  5 The test log loss is: 1.0264781602437116
```

4.2.2. Testing the model with best hyper paramters

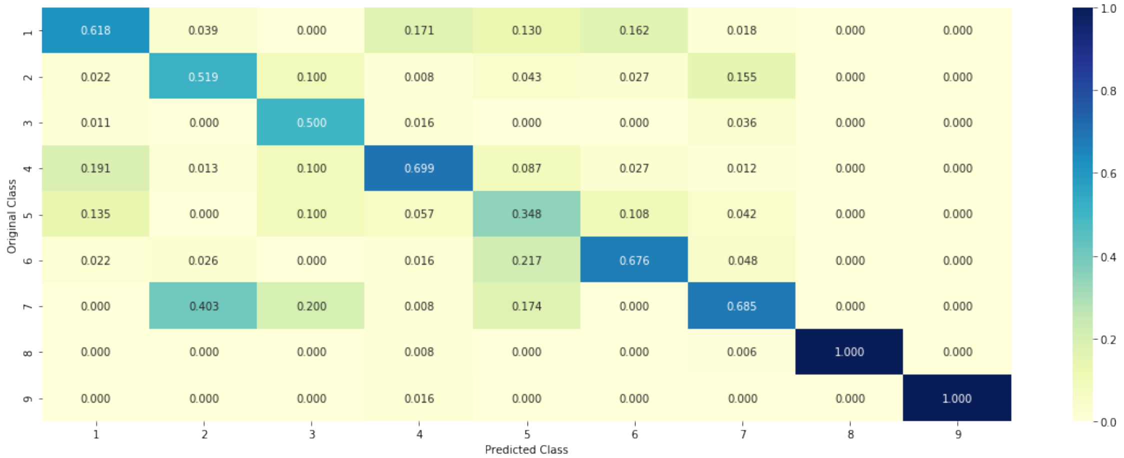
```
In [64]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
         # -------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
         # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

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

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding
```

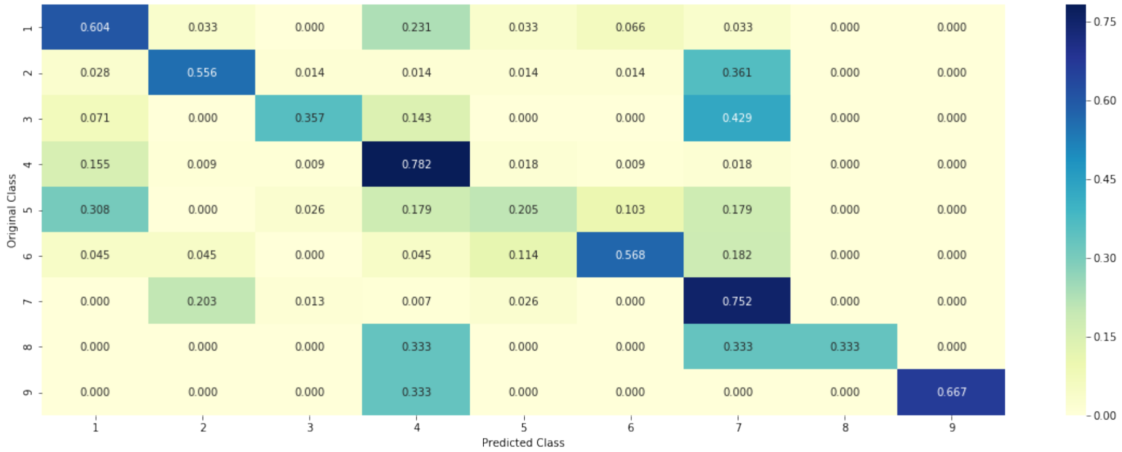
```
Log loss : 1.0046408472971262
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```



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



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



### 4.2.3.Sample Query point -1

```
In [65]: 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), al
         print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to clas
         print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

### 4.2.4. Sample Query Point-2

```
In [66]: 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 : 7
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [2 7
Fequency of nearest points : Counter({7: 4, 2: 1})
```

### 4.3. Logistic Regression

### 4.3.1. With Class balancing

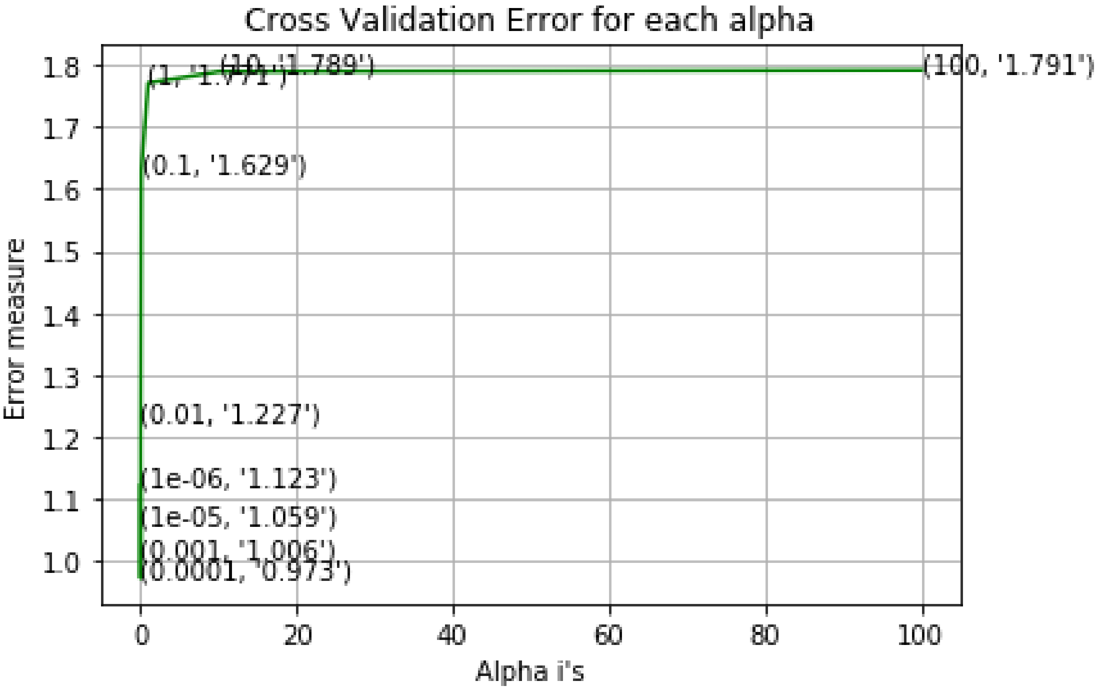
### 4.3.1.1. Hyper paramter tuning

```
In [67]: # 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 Gra
         # 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/module
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=3
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])       Fit the calibrated model
         # get_params([deep])       Get parameters for this estimator.
         # predict(X)       Predict the target of new samples.
         # predict_proba(X)       Posterior probabilities of classification
         #-------------------------------------
         # video link:
         #-------------------------------------

         alpha = [10 ** x for x in range(-6, 3)]
         print('alpha ',alpha)
         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=1
    # 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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
alpha  [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
for alpha = 1e-06
Log Loss : 1.1226145213962426
for alpha = 1e-05
Log Loss : 1.0592834881832796
for alpha = 0.0001
Log Loss : 0.9733629323289568
for alpha = 0.001
Log Loss : 1.0058288780621785
for alpha = 0.01
Log Loss : 1.2268252056735183
for alpha = 0.1
Log Loss : 1.6287446598805144
for alpha = 1
```

```
Log Loss : 1.771079017018445
for alpha = 10
Log Loss : 1.7889595897623696
for alpha = 100
Log Loss : 1.7911262162041959
```



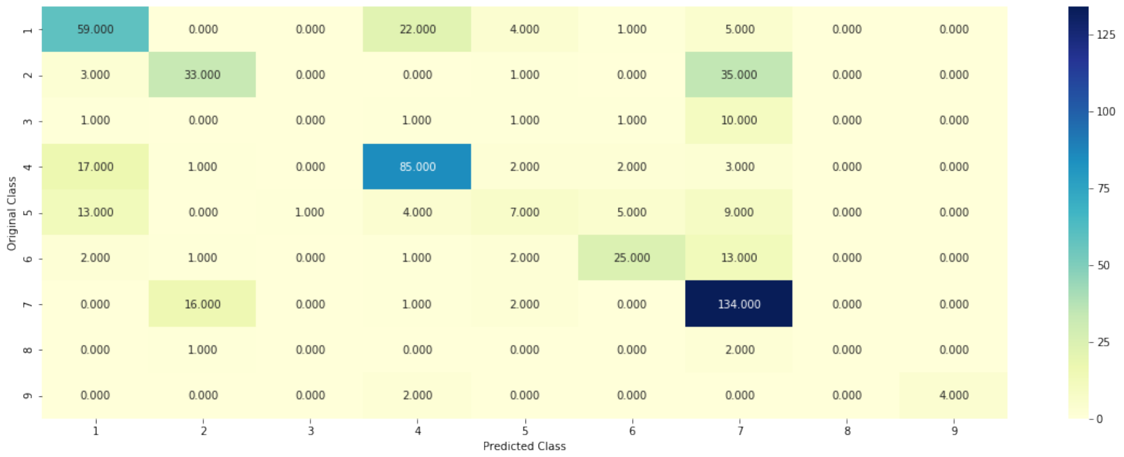
```
For values of best alpha =  0.0001 The train log loss is: 0.43916702890519291
For values of best alpha =  0.0001 The cross validation log loss is: 0.9733629323289568
For values of best alpha =  0.0001 The test log loss is: 0.9988417135791545
```

4.3.1.2. Testing the model with best hyper paramters

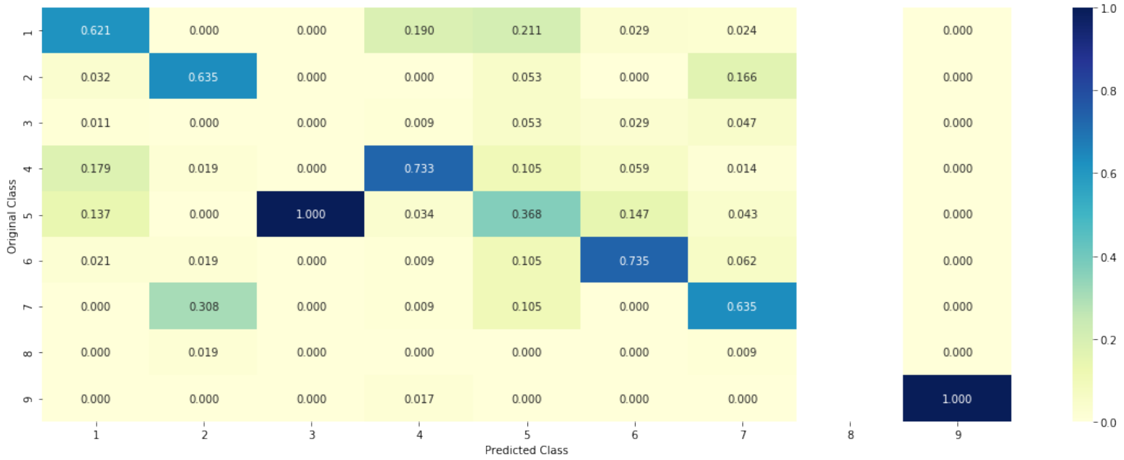
```
In [68]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, 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
```

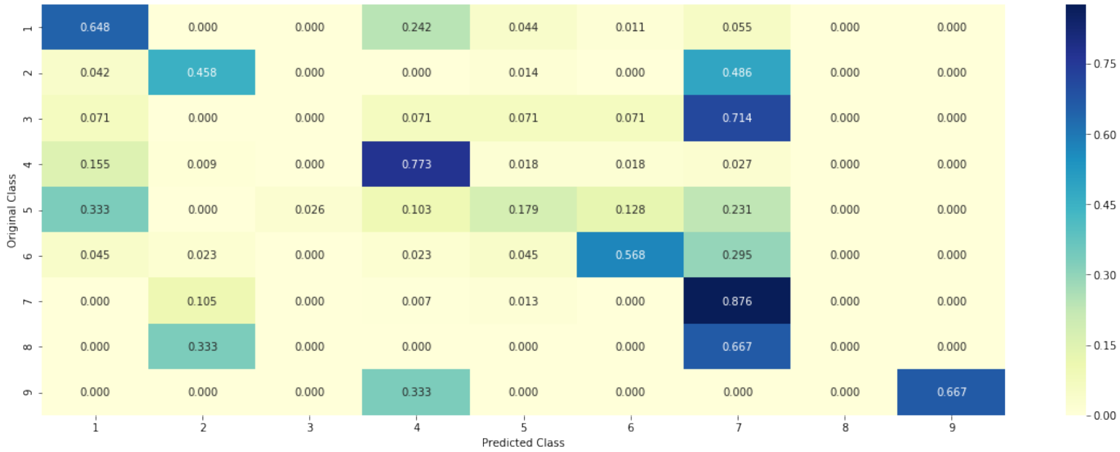
```
Log loss : 0.9733629323289568
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

```
In [69]: 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. Correctly Classified point

```
In [70]: # from tabulate import tabulate
         clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
         clf.fit(train_x_onehotCoding,train_y)
         test_point_index = 1
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2009 0.0208 0.0149 0.6981 0.0214 0.0165 0.0166 0.0046 0.006
Actual Class : 4
--------------------------------------------------
47 Text feature [mammalian] present in test data point [True]
50 Text feature [missense] present in test data point [True]
69 Text feature [tagged] present in test data point [True]
80 Text feature [suppressor] present in test data point [True]
114 Text feature [caused] present in test data point [True]
121 Text feature [tgf] present in test data point [True]
160 Text feature [show] present in test data point [True]
162 Text feature [protein] present in test data point [True]
166 Text feature [western] present in test data point [True]
171 Text feature [high] present in test data point [True]
172 Text feature [stability] present in test data point [True]
207 Text feature [age] present in test data point [True]
218 Text feature [tumorigenesis] present in test data point [True]
224 Text feature [germline] present in test data point [True]
237 Text feature [inactivation] present in test data point [True]
248 Text feature [suggesting] present in test data point [True]
256 Text feature [reduced] present in test data point [True]
271 Text feature [representative] present in test data point [True]
277 Text feature [buffer] present in test data point [True]
283 Text feature [short] present in test data point [True]
285 Text feature [represent] present in test data point [True]
290 Text feature [indicate] present in test data point [True]
306 Text feature [transfected] present in test data point [True]
311 Text feature [functional] present in test data point [True]
314 Text feature [resulting] present in test data point [True]
318 Text feature [bind] present in test data point [True]
328 Text feature [mice] present in test data point [True]
336 Text feature [due] present in test data point [True]
347 Text feature [4a] present in test data point [True]
350 Text feature [washed] present in test data point [True]
355 Text feature [localization] present in test data point [True]
360 Text feature [particular] present in test data point [True]
365 Text feature [ref] present in test data point [True]
373 Text feature [anti] present in test data point [True]
374 Text feature [often] present in test data point [True]
377 Text feature [regulation] present in test data point [True]
386 Text feature [activity] present in test data point [True]
393 Text feature [smad4] present in test data point [True]
407 Text feature [transfection] present in test data point [True]
```

```
408 Text feature [proteins] present in test data point [True]
413 Text feature [changes] present in test data point [True]
418 Text feature [loss] present in test data point [True]
427 Text feature [cases] present in test data point [True]
428 Text feature [cohort] present in test data point [True]
432 Text feature [bound] present in test data point [True]
433 Text feature [direct] present in test data point [True]
435 Text feature [genome] present in test data point [True]
448 Text feature [suggested] present in test data point [True]
450 Text feature [52] present in test data point [True]
451 Text feature [function] present in test data point [True]
455 Text feature [regions] present in test data point [True]
457 Text feature [family] present in test data point [True]
459 Text feature [characterized] present in test data point [True]
460 Text feature [comparison] present in test data point [True]
462 Text feature [lysates] present in test data point [True]
464 Text feature [27] present in test data point [True]
470 Text feature [flag] present in test data point [True]
473 Text feature [core] present in test data point [True]
476 Text feature [nuclear] present in test data point [True]
483 Text feature [deletion] present in test data point [True]
484 Text feature [despite] present in test data point [True]
488 Text feature [therefore] present in test data point [True]
491 Text feature [involved] present in test data point [True]
492 Text feature [times] present in test data point [True]
493 Text feature [recent] present in test data point [True]
494 Text feature [antibodies] present in test data point [True]
497 Text feature [44] present in test data point [True]
498 Text feature [29] present in test data point [True]
499 Text feature [altered] present in test data point [True]
Out of the top  500  features  69 are present in query point
```

#### 4.3.1.3.2. Incorrectly Classified point

```
In [71]: test_point_index = 100
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.003  0.0186 0.0022 0.0391 0.0029 0.0018 0.9298 0.0015 0.001
Actual Class : 7
```

```
--------------------------------------------------
12 Text feature [transformed] present in test data point [True]
13 Text feature [downstream] present in test data point [True]
33 Text feature [constitutive] present in test data point [True]
37 Text feature [activated] present in test data point [True]
38 Text feature [activation] present in test data point [True]
64 Text feature [61] present in test data point [True]
65 Text feature [codon] present in test data point [True]
74 Text feature [phosphorylated] present in test data point [True]
79 Text feature [examined] present in test data point [True]
85 Text feature [factor] present in test data point [True]
94 Text feature [promote] present in test data point [True]
148 Text feature [mapk] present in test data point [True]
151 Text feature [pathways] present in test data point [True]
166 Text feature [expressing] present in test data point [True]
177 Text feature [ba] present in test data point [True]
190 Text feature [signaling] present in test data point [True]
191 Text feature [nm] present in test data point [True]
208 Text feature [cultured] present in test data point [True]
211 Text feature [f3] present in test data point [True]
223 Text feature [regulated] present in test data point [True]
225 Text feature [enhanced] present in test data point [True]
257 Text feature [presence] present in test data point [True]
294 Text feature [phosphorylation] present in test data point [True]
306 Text feature [2a] present in test data point [True]
309 Text feature [bone] present in test data point [True]
314 Text feature [phospho] present in test data point [True]
320 Text feature [inhibitor] present in test data point [True]
336 Text feature [positive] present in test data point [True]
348 Text feature [marrow] present in test data point [True]
381 Text feature [extracellular] present in test data point [True]
388 Text feature [high] present in test data point [True]
389 Text feature [treated] present in test data point [True]
391 Text feature [factors] present in test data point [True]
400 Text feature [coding] present in test data point [True]
404 Text feature [university] present in test data point [True]
420 Text feature [2001] present in test data point [True]
422 Text feature [tissue] present in test data point [True]
427 Text feature [observations] present in test data point [True]
438 Text feature [survival] present in test data point [True]
439 Text feature [leukemia] present in test data point [True]
441 Text feature [2b] present in test data point [True]
444 Text feature [cdna] present in test data point [True]
467 Text feature [interestingly] present in test data point [True]
477 Text feature [fold] present in test data point [True]
Out of the top  500  features  44 are present in query point
```

### 4.3.2. Without Class balancing

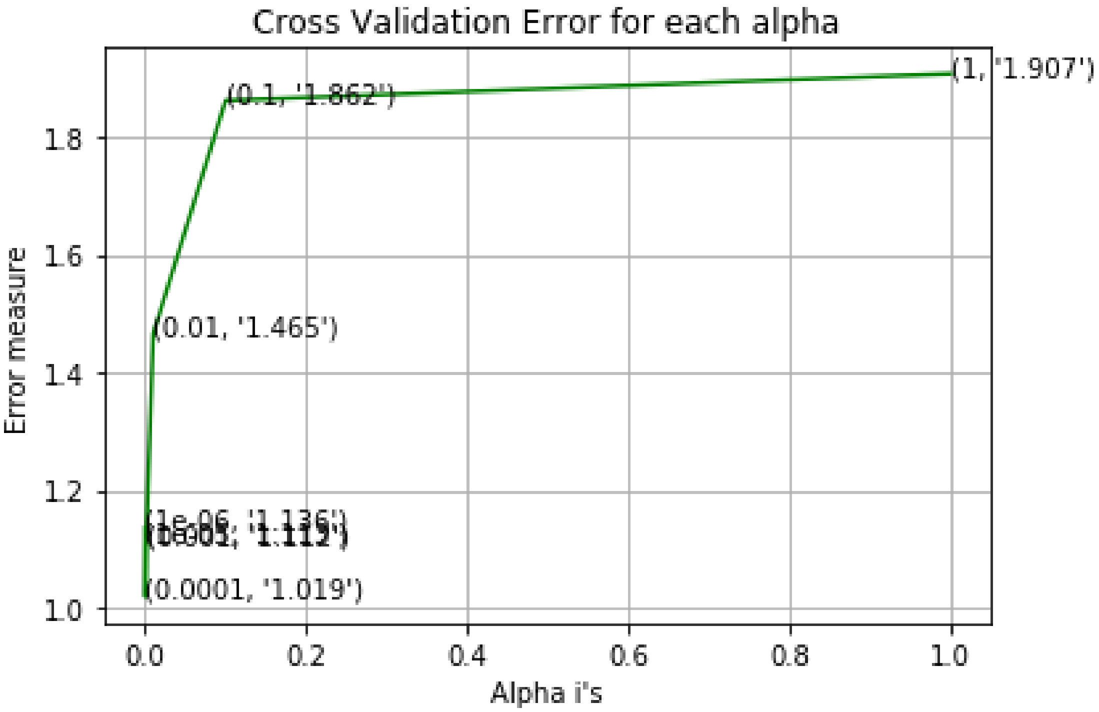
#### 4.3.2.1. Hyper paramter tuning

In [72]:

```
# 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=4
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

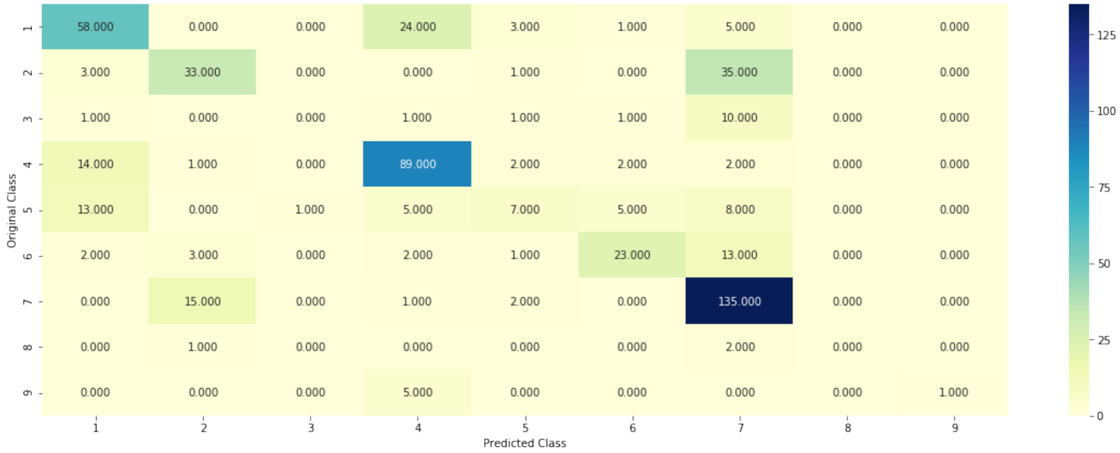
```
for alpha = 1e-06
Log Loss : 1.1361780246678814
for alpha = 1e-05
Log Loss : 1.1127014922803464
for alpha = 0.0001
Log Loss : 1.019022238539661
for alpha = 0.001
Log Loss : 1.1116116790283928
for alpha = 0.01
Log Loss : 1.4654157548103004
for alpha = 0.1
Log Loss : 1.8616038845694647
for alpha = 1
Log Loss : 1.9067955301269583
```



```
For values of best alpha =  0.0001 The train log loss is: 0.42938512770897397
For values of best alpha =  0.0001 The cross validation log loss is: 1.019022238539661
For values of best alpha =  0.0001 The test log loss is: 1.0258545930382228
```

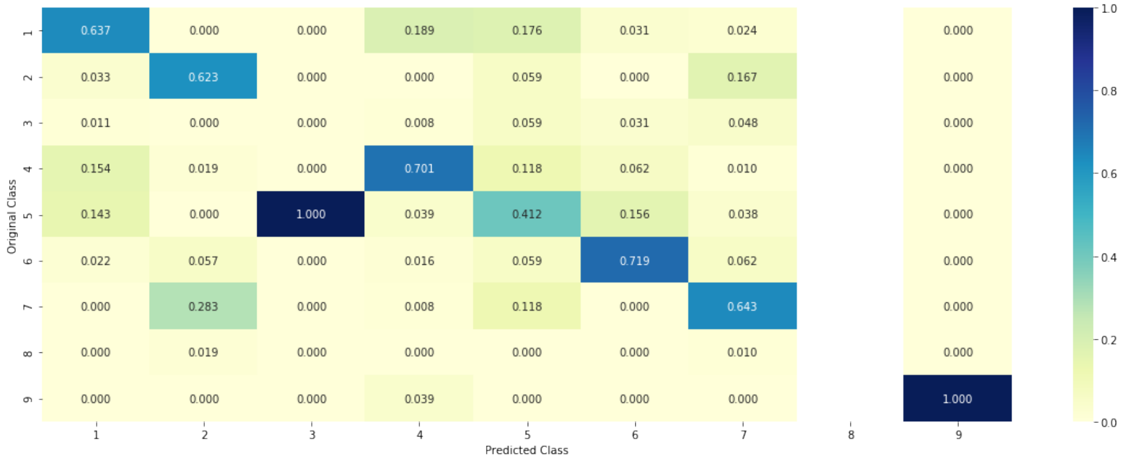
4.3.2.2. Testing model with best hyper parameters

```
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:
         #------------------------------

         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, c
Log loss : 1.019022238539661
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



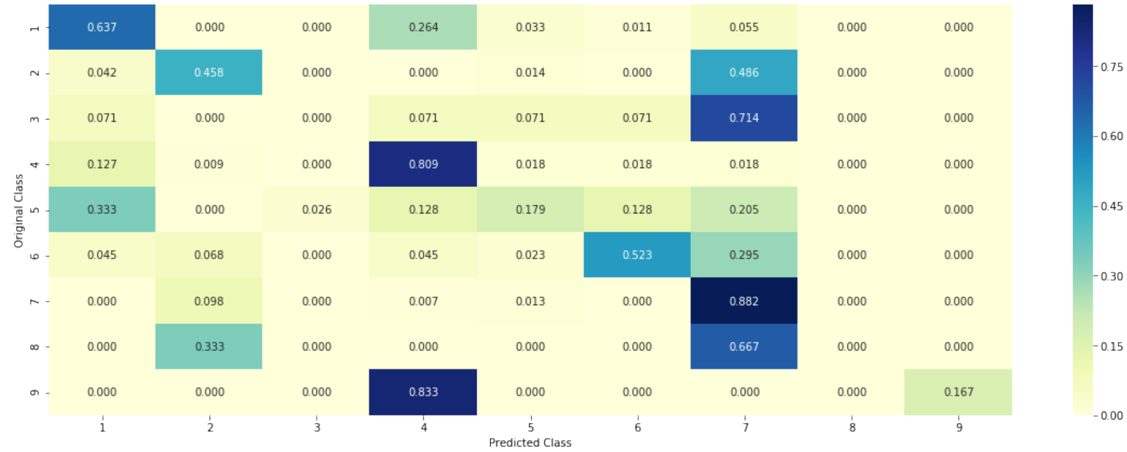
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 58.000 | 0.000 | 0.000 | 24.000 | 3.000 | 1.000 | 5.000 | 0.000 | 0.000 |
| 2 | 3.000 | 33.000 | 0.000 | 0.000 | 1.000 | 0.000 | 35.000 | 0.000 | 0.000 |
| 3 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 10.000 | 0.000 | 0.000 |
| 4 | 14.000 | 1.000 | 0.000 | 89.000 | 2.000 | 2.000 | 2.000 | 0.000 | 0.000 |
| 5 | 13.000 | 0.000 | 1.000 | 5.000 | 7.000 | 5.000 | 8.000 | 0.000 | 0.000 |
| 6 | 2.000 | 3.000 | 0.000 | 2.000 | 1.000 | 23.000 | 13.000 | 0.000 | 0.000 |
| 7 | 0.000 | 15.000 | 0.000 | 1.000 | 2.000 | 0.000 | 135.000 | 0.000 | 0.000 |
| 8 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 2.000 | 0.000 | 0.000 |
| 9 | 0.000 | 0.000 | 0.000 | 5.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |

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



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.637 | 0.000 | 0.000 | 0.189 | 0.176 | 0.031 | 0.024 | | 0.000 |
| 2 | 0.033 | 0.623 | 0.000 | 0.000 | 0.059 | 0.000 | 0.167 | | 0.000 |
| 3 | 0.011 | 0.000 | 0.000 | 0.008 | 0.059 | 0.031 | 0.048 | | 0.000 |
| 4 | 0.154 | 0.019 | 0.000 | 0.701 | 0.118 | 0.062 | 0.010 | | 0.000 |
| 5 | 0.143 | 0.000 | 1.000 | 0.039 | 0.412 | 0.156 | 0.038 | | 0.000 |
| 6 | 0.022 | 0.057 | 0.000 | 0.016 | 0.059 | 0.719 | 0.062 | | 0.000 |
| 7 | 0.000 | 0.283 | 0.000 | 0.008 | 0.118 | 0.000 | 0.643 | | 0.000 |
| 8 | 0.000 | 0.019 | 0.000 | 0.000 | 0.000 | 0.000 | 0.010 | | 0.000 |
| 9 | 0.000 | 0.000 | 0.000 | 0.039 | 0.000 | 0.000 | 0.000 | | 1.000 |

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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [74]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         clf.fit(train_x_onehotCoding,train_y)
         test_point_index = 1
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2046 0.0209 0.0165 0.6978 0.0205 0.016  0.0174 0.0025 0.003
Actual Class : 4
--------------------------------------------------
59 Text feature [mammalian] present in test data point [True]
80 Text feature [missense] present in test data point [True]
85 Text feature [suppressor] present in test data point [True]
88 Text feature [tagged] present in test data point [True]
99 Text feature [caused] present in test data point [True]
131 Text feature [tgf] present in test data point [True]
156 Text feature [western] present in test data point [True]
166 Text feature [show] present in test data point [True]
174 Text feature [high] present in test data point [True]
177 Text feature [protein] present in test data point [True]
192 Text feature [tumorigenesis] present in test data point [True]
206 Text feature [stability] present in test data point [True]
217 Text feature [age] present in test data point [True]
246 Text feature [suggesting] present in test data point [True]
```

```
258 Text feature [germline] present in test data point [True]
264 Text feature [representative] present in test data point [True]
268 Text feature [represent] present in test data point [True]
270 Text feature [inactivation] present in test data point [True]
276 Text feature [reduced] present in test data point [True]
278 Text feature [bind] present in test data point [True]
302 Text feature [short] present in test data point [True]
303 Text feature [indicate] present in test data point [True]
304 Text feature [often] present in test data point [True]
307 Text feature [functional] present in test data point [True]
308 Text feature [resulting] present in test data point [True]
324 Text feature [buffer] present in test data point [True]
327 Text feature [transfected] present in test data point [True]
332 Text feature [4a] present in test data point [True]
334 Text feature [due] present in test data point [True]
336 Text feature [52] present in test data point [True]
347 Text feature [washed] present in test data point [True]
350 Text feature [anti] present in test data point [True]
352 Text feature [mice] present in test data point [True]
364 Text feature [regions] present in test data point [True]
365 Text feature [particular] present in test data point [True]
370 Text feature [regulation] present in test data point [True]
372 Text feature [localization] present in test data point [True]
376 Text feature [activity] present in test data point [True]
388 Text feature [smad4] present in test data point [True]
393 Text feature [cohort] present in test data point [True]
395 Text feature [ref] present in test data point [True]
401 Text feature [bound] present in test data point [True]
405 Text feature [cases] present in test data point [True]
406 Text feature [transfection] present in test data point [True]
410 Text feature [direct] present in test data point [True]
414 Text feature [proteins] present in test data point [True]
417 Text feature [lysates] present in test data point [True]
419 Text feature [changes] present in test data point [True]
420 Text feature [function] present in test data point [True]
423 Text feature [44] present in test data point [True]
440 Text feature [despite] present in test data point [True]
441 Text feature [therefore] present in test data point [True]
446 Text feature [genome] present in test data point [True]
447 Text feature [characterized] present in test data point [True]
448 Text feature [45] present in test data point [True]
450 Text feature [recent] present in test data point [True]
453 Text feature [suggested] present in test data point [True]
457 Text feature [27] present in test data point [True]
458 Text feature [loss] present in test data point [True]
465 Text feature [antibodies] present in test data point [True]
469 Text feature [comparison] present in test data point [True]
470 Text feature [core] present in test data point [True]
```

```
474 Text feature [genomic] present in test data point [True]
477 Text feature [times] present in test data point [True]
478 Text feature [29] present in test data point [True]
479 Text feature [family] present in test data point [True]
480 Text feature [determine] present in test data point [True]
481 Text feature [suggest] present in test data point [True]
482 Text feature [26] present in test data point [True]
487 Text feature [still] present in test data point [True]
488 Text feature [flag] present in test data point [True]
489 Text feature [42] present in test data point [True]
492 Text feature [considered] present in test data point [True]
493 Text feature [ha] present in test data point [True]
494 Text feature [differences] present in test data point [True]
495 Text feature [involved] present in test data point [True]
Out of the top  500  features  76 are present in query point
```

#### 4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [75]: test_point_index = 100
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.300e-03 1.810e-02 2.500e-03 4.550e-02 2.500e-03 1.800e-03 9
  3.000e-04 2.000e-04]]
Actual Class : 7
--------------------------------------------------
19 Text feature [downstream] present in test data point [True]
20 Text feature [transformed] present in test data point [True]
66 Text feature [activated] present in test data point [True]
81 Text feature [codon] present in test data point [True]
82 Text feature [examined] present in test data point [True]
83 Text feature [activation] present in test data point [True]
84 Text feature [61] present in test data point [True]
88 Text feature [phosphorylated] present in test data point [True]
96 Text feature [constitutive] present in test data point [True]
107 Text feature [factor] present in test data point [True]
142 Text feature [promote] present in test data point [True]
186 Text feature [expressing] present in test data point [True]
190 Text feature [ba] present in test data point [True]
222 Text feature [pathways] present in test data point [True]
```

```
232 Text feature [f3] present in test data point [True]
249 Text feature [enhanced] present in test data point [True]
252 Text feature [nm] present in test data point [True]
270 Text feature [signaling] present in test data point [True]
287 Text feature [2a] present in test data point [True]
288 Text feature [presence] present in test data point [True]
302 Text feature [cultured] present in test data point [True]
305 Text feature [mapk] present in test data point [True]
320 Text feature [coding] present in test data point [True]
324 Text feature [positive] present in test data point [True]
327 Text feature [regulated] present in test data point [True]
334 Text feature [phosphorylation] present in test data point [True]
359 Text feature [bone] present in test data point [True]
360 Text feature [2001] present in test data point [True]
362 Text feature [observations] present in test data point [True]
368 Text feature [high] present in test data point [True]
385 Text feature [factors] present in test data point [True]
407 Text feature [phospho] present in test data point [True]
408 Text feature [inhibitor] present in test data point [True]
414 Text feature [university] present in test data point [True]
430 Text feature [2b] present in test data point [True]
436 Text feature [extracellular] present in test data point [True]
441 Text feature [fold] present in test data point [True]
442 Text feature [exon] present in test data point [True]
450 Text feature [treated] present in test data point [True]
459 Text feature [marrow] present in test data point [True]
469 Text feature [membrane] present in test data point [True]
485 Text feature [survival] present in test data point [True]
487 Text feature [tissue] present in test data point [True]
Out of the top  500  features  43 are present in query point
```

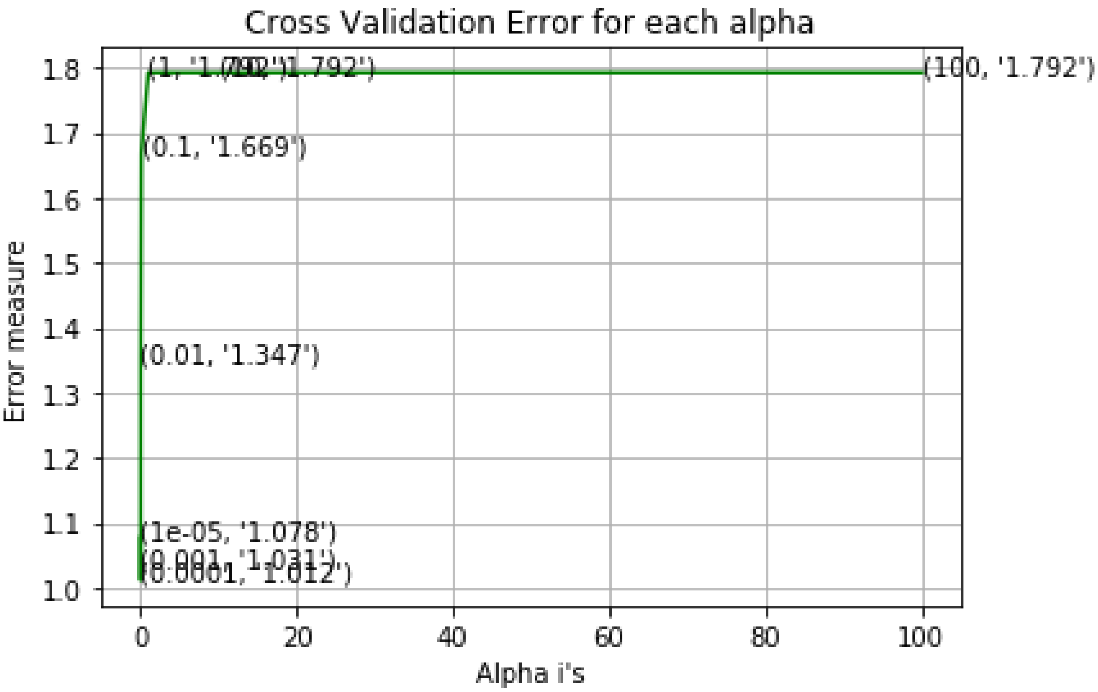
### 4.4. Linear Support Vector Machines

### 4.4.1. Hyper paramter tuning

```
In [76]: # read more about support vector machines with linear kernals here http://scikit-lear

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

         # Some of methods of SVM()
         # fit(X, y, [sample_weight])        Fit the SVM model according to the given training
         # predict(X)        Perform classification on samples in X.
         # --------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
         # --------------------------------
         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])        Fit the calibrated model
         # get_params([deep])        Get parameters for this estimator.
         # predict(X)        Predict the target of new samples.
         # predict_proba(X)        Posterior probabilities of classification
         #-------------------------------------
         # video link:
         #-------------------------------------

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

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


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

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

```
for C = 1e-05
Log Loss : 1.077848175029197
for C = 0.0001
Log Loss : 1.0122200442734932
for C = 0.001
Log Loss : 1.0313244189991897
for C = 0.01
Log Loss : 1.346879225564524
for C = 0.1
Log Loss : 1.6687268229384513
for C = 1
Log Loss : 1.791770054017661
for C = 10
Log Loss : 1.7917640081207253
for C = 100
Log Loss : 1.7917640697580601
```

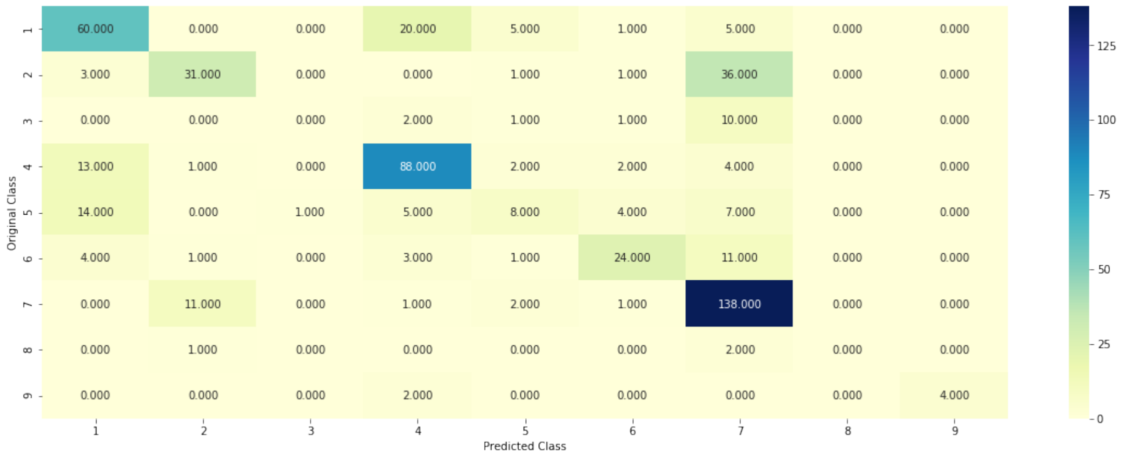


```
For values of best alpha =  0.0001 The train log loss is: 0.47877678335465024
For values of best alpha =  0.0001 The cross validation log loss is: 1.0122200442734932
For values of best alpha =  0.0001 The test log loss is: 1.0555794392942766
```

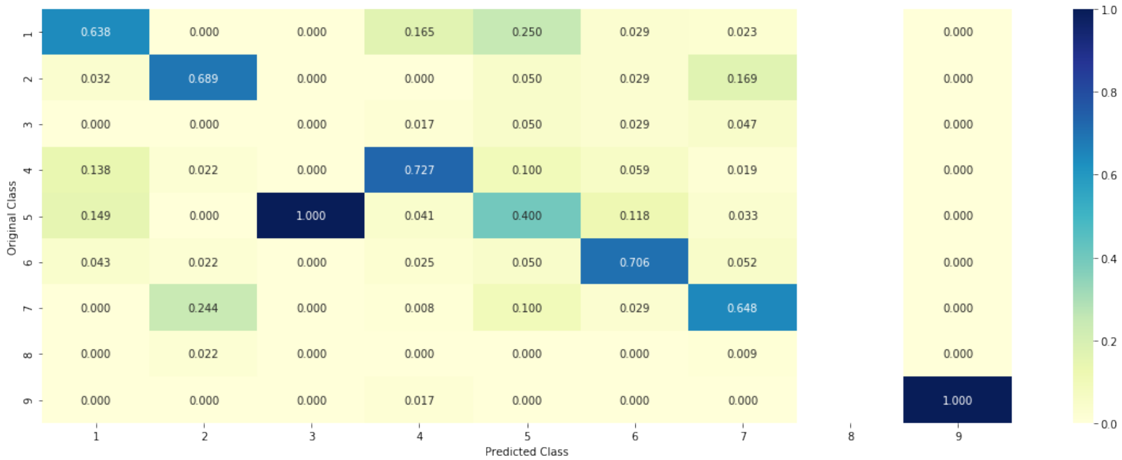
### 4.4.2. Testing model with best hyper parameters

```
In [77]: # 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_
```

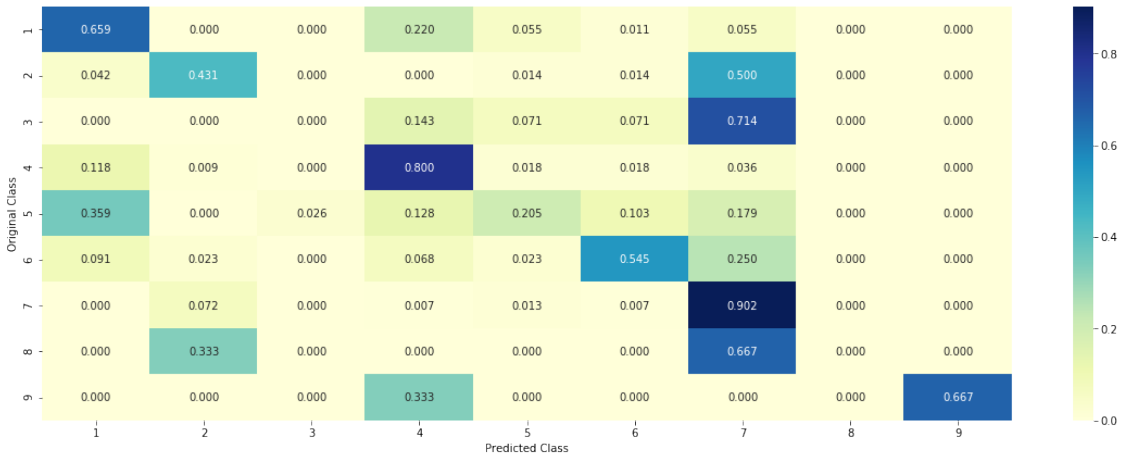
```
Log loss : 1.0122200442734932
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```



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



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



### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

```
In [78]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
         clf.fit(train_x_onehotCoding,train_y)
         test_point_index = 1
         # test_point_index = 100
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
        print("-"*50)
        get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2326 0.0341 0.0232 0.6021 0.0275 0.0232 0.047  0.0039 0.006
Actual Class : 4
--------------------------------------------------
76 Text feature [missense] present in test data point [True]
78 Text feature [mammalian] present in test data point [True]
79 Text feature [tgf] present in test data point [True]
202 Text feature [caused] present in test data point [True]
204 Text feature [show] present in test data point [True]
205 Text feature [suppressor] present in test data point [True]
209 Text feature [tagged] present in test data point [True]
210 Text feature [suggesting] present in test data point [True]
211 Text feature [high] present in test data point [True]
213 Text feature [age] present in test data point [True]
214 Text feature [short] present in test data point [True]
217 Text feature [anti] present in test data point [True]
219 Text feature [protein] present in test data point [True]
220 Text feature [germline] present in test data point [True]
222 Text feature [due] present in test data point [True]
224 Text feature [bind] present in test data point [True]
225 Text feature [52] present in test data point [True]
229 Text feature [often] present in test data point [True]
231 Text feature [functional] present in test data point [True]
232 Text feature [stability] present in test data point [True]
233 Text feature [western] present in test data point [True]
234 Text feature [reduced] present in test data point [True]
235 Text feature [tumorigenesis] present in test data point [True]
237 Text feature [regions] present in test data point [True]
242 Text feature [45] present in test data point [True]
244 Text feature [mice] present in test data point [True]
246 Text feature [activated] present in test data point [True]
247 Text feature [3b] present in test data point [True]
249 Text feature [resulting] present in test data point [True]
251 Text feature [represent] present in test data point [True]
252 Text feature [bound] present in test data point [True]
253 Text feature [determine] present in test data point [True]
254 Text feature [cases] present in test data point [True]
255 Text feature [buffer] present in test data point [True]
257 Text feature [ref] present in test data point [True]
258 Text feature [representative] present in test data point [True]
458 Text feature [washed] present in test data point [True]
459 Text feature [indicate] present in test data point [True]
463 Text feature [4a] present in test data point [True]
466 Text feature [differences] present in test data point [True]
468 Text feature [proteins] present in test data point [True]
```

```
469 Text feature [despite] present in test data point [True]
472 Text feature [transfection] present in test data point [True]
473 Text feature [inactivation] present in test data point [True]
476 Text feature [hours] present in test data point [True]
477 Text feature [regulation] present in test data point [True]
478 Text feature [family] present in test data point [True]
481 Text feature [particular] present in test data point [True]
484 Text feature [genomic] present in test data point [True]
485 Text feature [29] present in test data point [True]
487 Text feature [mouse] present in test data point [True]
489 Text feature [smad4] present in test data point [True]
490 Text feature [12] present in test data point [True]
491 Text feature [mechanisms] present in test data point [True]
492 Text feature [amino] present in test data point [True]
493 Text feature [localization] present in test data point [True]
495 Text feature [lower] present in test data point [True]
496 Text feature [44] present in test data point [True]
497 Text feature [42] present in test data point [True]
498 Text feature [comparison] present in test data point [True]
499 Text feature [suggested] present in test data point [True]
Out of the top  500  features  61 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
In [79]: test_point_index = 100
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.004  0.0297 0.0047 0.1028 0.0162 0.0051 0.8349 0.0011 0.0015
Actual Class : 7
--------------------------------------------------
29 Text feature [transformed] present in test data point [True]
30 Text feature [downstream] present in test data point [True]
33 Text feature [phosphorylated] present in test data point [True]
232 Text feature [61] present in test data point [True]
233 Text feature [examined] present in test data point [True]
234 Text feature [activated] present in test data point [True]
239 Text feature [presence] present in test data point [True]
244 Text feature [factor] present in test data point [True]
245 Text feature [nm] present in test data point [True]
```

```
248 Text feature [expressing] present in test data point [True]
251 Text feature [ba] present in test data point [True]
253 Text feature [pathways] present in test data point [True]
257 Text feature [mapk] present in test data point [True]
258 Text feature [f3] present in test data point [True]
262 Text feature [activation] present in test data point [True]
265 Text feature [observations] present in test data point [True]
267 Text feature [2001] present in test data point [True]
268 Text feature [university] present in test data point [True]
271 Text feature [phospho] present in test data point [True]
272 Text feature [fold] present in test data point [True]
275 Text feature [positive] present in test data point [True]
277 Text feature [clinically] present in test data point [True]
278 Text feature [coding] present in test data point [True]
281 Text feature [high] present in test data point [True]
282 Text feature [01] present in test data point [True]
284 Text feature [codon] present in test data point [True]
285 Text feature [factors] present in test data point [True]
286 Text feature [lead] present in test data point [True]
288 Text feature [phosphorylation] present in test data point [True]
291 Text feature [enhanced] present in test data point [True]
292 Text feature [signaling] present in test data point [True]
293 Text feature [found] present in test data point [True]
294 Text feature [materials] present in test data point [True]
295 Text feature [regulated] present in test data point [True]
296 Text feature [phosphate] present in test data point [True]
300 Text feature [tissues] present in test data point [True]
302 Text feature [promote] present in test data point [True]
304 Text feature [14] present in test data point [True]
308 Text feature [exon] present in test data point [True]
310 Text feature [constitutive] present in test data point [True]
312 Text feature [cdna] present in test data point [True]
314 Text feature [mechanisms] present in test data point [True]
315 Text feature [wt] present in test data point [True]
318 Text feature [treated] present in test data point [True]
319 Text feature [cultured] present in test data point [True]
321 Text feature [increase] present in test data point [True]
Out of the top  500  features  46 are present in query point
```

### 4.5 Random Forest Classifier

### 4.5.1. Hyper paramter tuning (With One hot Encoding)

```
In [80]: # --------------------------------
         # 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 los
         predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
         print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validat
         predict_y = sig_clf.predict_proba(test_x_onehotCoding)
         print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
```

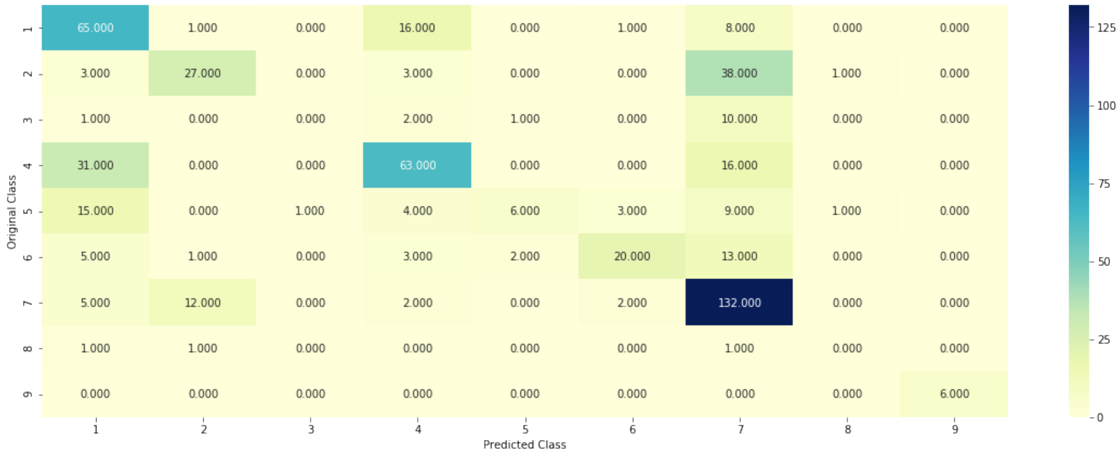
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1946146355042506
for n_estimators = 100 and max depth =  10
Log Loss : 1.2546815993659284
for n_estimators = 200 and max depth =  5
Log Loss : 1.185191504867985
for n_estimators = 200 and max depth =  10
Log Loss : 1.2371062243138797
for n_estimators = 500 and max depth =  5
Log Loss : 1.1825153586877637
for n_estimators = 500 and max depth =  10
Log Loss : 1.239022076656428
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1769664621965408
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2323966853020778
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1775684947131833
for n_estimators = 2000 and max depth =  10
Log Loss : 1.234035839976541
For values of best estimator =  1000 The train log loss is: 0.8527403405228027
For values of best estimator =  1000 The cross validation log loss is: 1.176966462196541
For values of best estimator =  1000 The test log loss is: 1.174699498033692
```

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

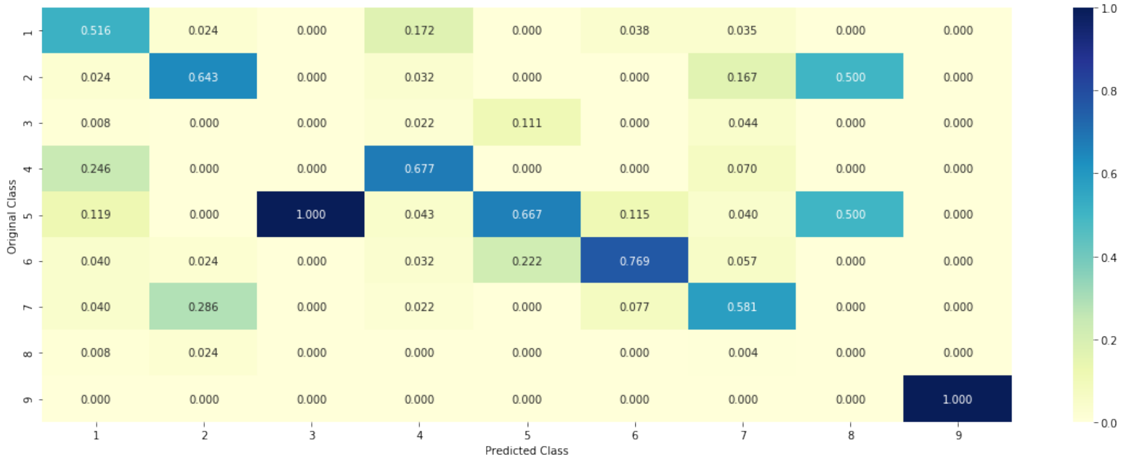
```
In [81]: # --------------------------------
```

```
# 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_
```

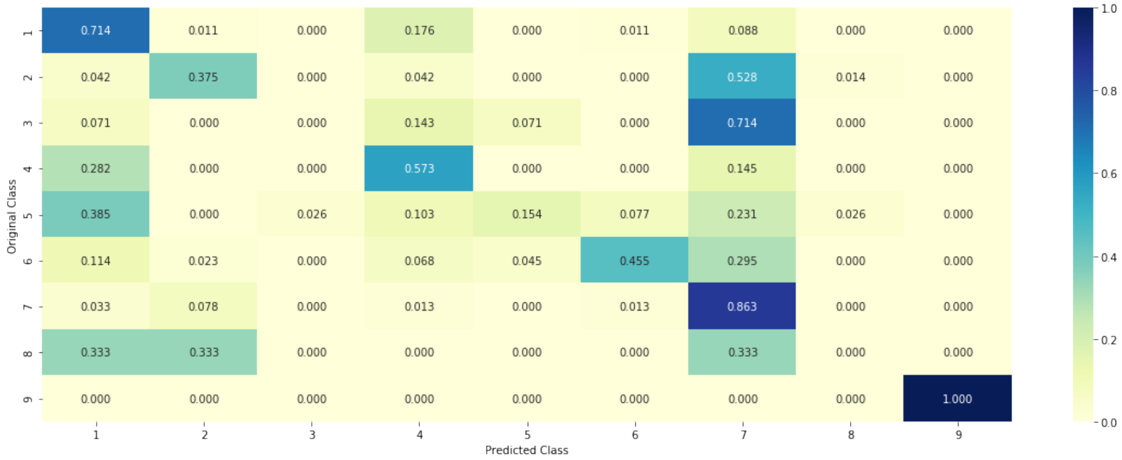
```
Log loss : 1.176966462196541
Number of mis-classified points : 0.40037593984962405
-------------------- Confusion matrix --------------------
```



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



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



### 4.5.3. Feature Importance

### 4.5.3.1. Correctly Classified point

```
In [82]: # test_point_index = 10
         clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
         clf.fit(train_x_onehotCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_onehotCoding, train_y)

         test_point_index = 1
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
```

```
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.feature_importances_)
        print("-"*50)
        get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.272  0.032  0.0182 0.524  0.048  0.0445 0.0468 0.0047 0.0098
Actual Class : 4
--------------------------------------------------
3 Text feature [function] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [activated] present in test data point [True]
6 Text feature [suppressor] present in test data point [True]
7 Text feature [activation] present in test data point [True]
9 Text feature [phosphorylation] present in test data point [True]
10 Text feature [loss] present in test data point [True]
11 Text feature [missense] present in test data point [True]
13 Text feature [inhibitor] present in test data point [True]
17 Text feature [constitutively] present in test data point [True]
20 Text feature [receptor] present in test data point [True]
27 Text feature [protein] present in test data point [True]
29 Text feature [stability] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [classified] present in test data point [True]
35 Text feature [functional] present in test data point [True]
38 Text feature [signaling] present in test data point [True]
39 Text feature [pathogenic] present in test data point [True]
46 Text feature [cells] present in test data point [True]
49 Text feature [cell] present in test data point [True]
54 Text feature [mammalian] present in test data point [True]
61 Text feature [expression] present in test data point [True]
62 Text feature [proteins] present in test data point [True]
63 Text feature [patients] present in test data point [True]
66 Text feature [treated] present in test data point [True]
68 Text feature [information] present in test data point [True]
69 Text feature [predicted] present in test data point [True]
72 Text feature [inactivation] present in test data point [True]
80 Text feature [advanced] present in test data point [True]
82 Text feature [activity] present in test data point [True]
86 Text feature [use] present in test data point [True]
87 Text feature [clinical] present in test data point [True]
90 Text feature [lines] present in test data point [True]
91 Text feature [factor] present in test data point [True]
93 Text feature [splice] present in test data point [True]
94 Text feature [active] present in test data point [True]
97 Text feature [dna] present in test data point [True]
Out of the top  100  features  37 are present in query point
```

### 4.5.3.2. Inorrectly Classified point

```
In [83]: test_point_index = 100
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actuall Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0594 0.1223 0.0231 0.0655 0.0488 0.0372 0.6336 0.006  0.004
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [function] present in test data point [True]
5 Text feature [activated] present in test data point [True]
7 Text feature [activation] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
9 Text feature [phosphorylation] present in test data point [True]
11 Text feature [missense] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [inhibitor] present in test data point [True]
27 Text feature [protein] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [classified] present in test data point [True]
35 Text feature [functional] present in test data point [True]
36 Text feature [downstream] present in test data point [True]
37 Text feature [ba] present in test data point [True]
38 Text feature [signaling] present in test data point [True]
39 Text feature [pathogenic] present in test data point [True]
40 Text feature [f3] present in test data point [True]
42 Text feature [erk] present in test data point [True]
43 Text feature [months] present in test data point [True]
44 Text feature [extracellular] present in test data point [True]
46 Text feature [cells] present in test data point [True]
49 Text feature [cell] present in test data point [True]
54 Text feature [mammalian] present in test data point [True]
61 Text feature [expression] present in test data point [True]
62 Text feature [proteins] present in test data point [True]
63 Text feature [patients] present in test data point [True]
65 Text feature [phosphatase] present in test data point [True]
66 Text feature [treated] present in test data point [True]
```

```
75 Text feature [mapk] present in test data point [True]
77 Text feature [classification] present in test data point [True]
79 Text feature [expressing] present in test data point [True]
81 Text feature [proliferation] present in test data point [True]
82 Text feature [activity] present in test data point [True]
83 Text feature [affected] present in test data point [True]
84 Text feature [phospho] present in test data point [True]
85 Text feature [ras] present in test data point [True]
86 Text feature [use] present in test data point [True]
87 Text feature [clinical] present in test data point [True]
90 Text feature [lines] present in test data point [True]
91 Text feature [factor] present in test data point [True]
94 Text feature [active] present in test data point [True]
96 Text feature [survival] present in test data point [True]
97 Text feature [dna] present in test data point [True]
Out of the top  100  features  45 are present in query point
```

### 4.5.3. Hyper paramter tuning (With Response Coding)

```
In [84]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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


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

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

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

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

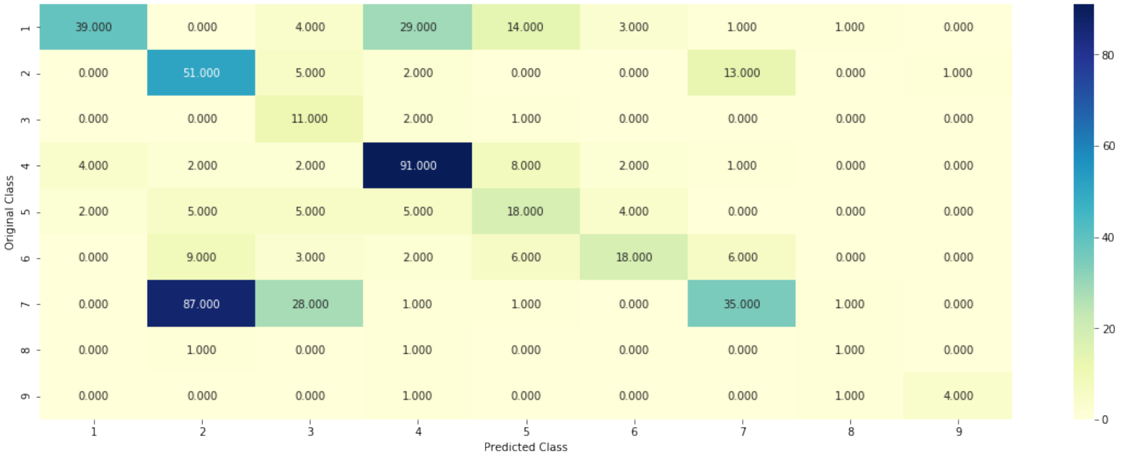
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1481377816860823
```

```
for n_estimators = 10 and max depth =  3
Log Loss : 1.7199970157910647
for n_estimators = 10 and max depth =  5
Log Loss : 1.5399153848549174
for n_estimators = 10 and max depth =  10
Log Loss : 1.6949864970992254
for n_estimators = 50 and max depth =  2
Log Loss : 1.6841251906601102
for n_estimators = 50 and max depth =  3
Log Loss : 1.4029288008624021
for n_estimators = 50 and max depth =  5
Log Loss : 1.2877280837142862
for n_estimators = 50 and max depth =  10
Log Loss : 1.6348110584416888
for n_estimators = 100 and max depth =  2
Log Loss : 1.563840924795056
for n_estimators = 100 and max depth =  3
Log Loss : 1.4575316493591823
for n_estimators = 100 and max depth =  5
Log Loss : 1.2865001834174978
for n_estimators = 100 and max depth =  10
Log Loss : 1.5917238299264034
for n_estimators = 200 and max depth =  2
Log Loss : 1.5945038918227343
for n_estimators = 200 and max depth =  3
Log Loss : 1.4850263331175053
for n_estimators = 200 and max depth =  5
Log Loss : 1.3033907077775777
for n_estimators = 200 and max depth =  10
Log Loss : 1.6119391844802033
for n_estimators = 500 and max depth =  2
Log Loss : 1.6567231415727905
for n_estimators = 500 and max depth =  3
Log Loss : 1.5014204746331525
for n_estimators = 500 and max depth =  5
Log Loss : 1.3193633224875094
for n_estimators = 500 and max depth =  10
Log Loss : 1.5984680369342452
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6264649943982123
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5212167781468002
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3175892392072095
for n_estimators = 1000 and max depth =  10
Log Loss : 1.5984945627666398
For values of best alpha =  100 The train log loss is: 0.054083920108801836
For values of best alpha =  100 The cross validation log loss is: 1.286500183417498
For values of best alpha =  100 The test log loss is: 1.2894477844824646
```

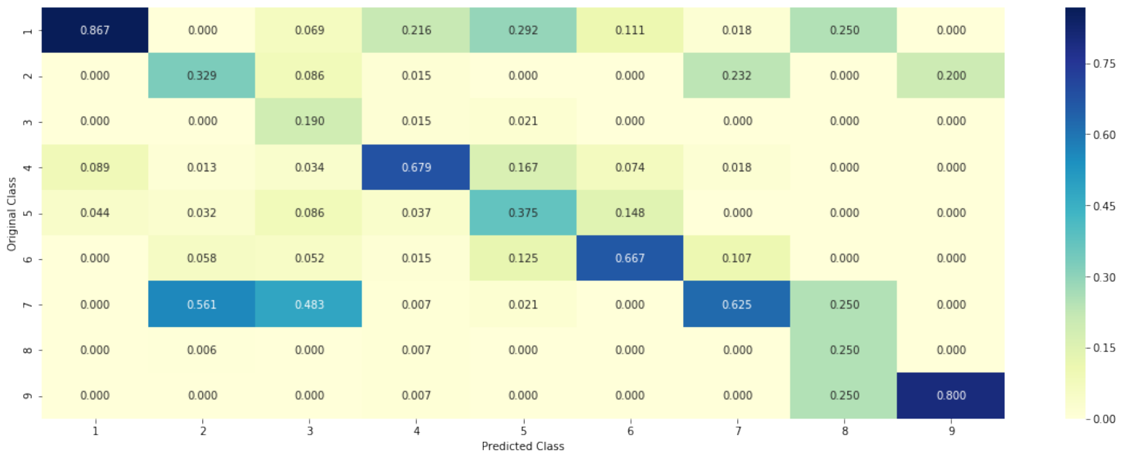
4.5.4. Testing model with best hyper parameters (Response Coding)

```
In [85]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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

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

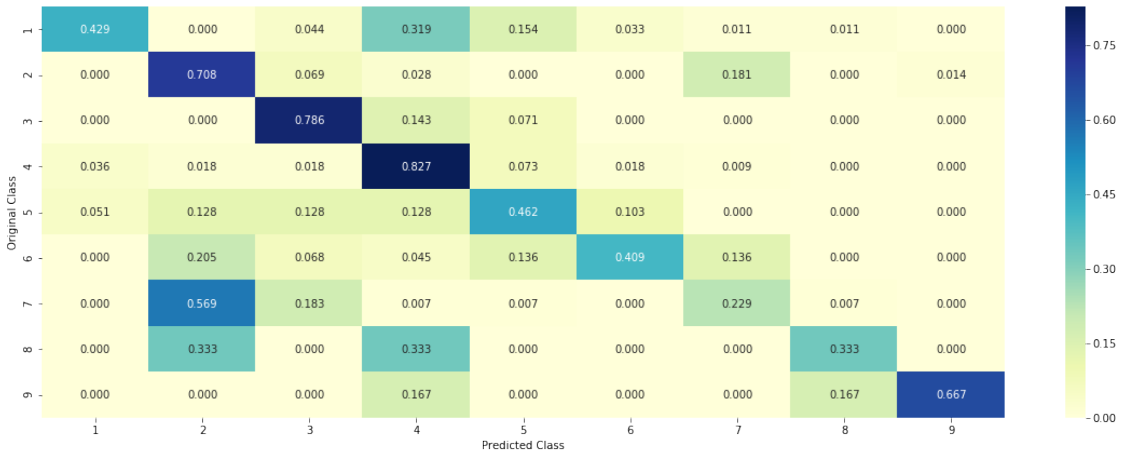
```
Log loss : 1.2865001834164913
Number of mis-classified points : 0.49624060150375937
-------------------- Confusion matrix --------------------
```



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



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



### 4.5.5. Feature Importance

#### 4.5.5.1. Correctly Classified point

```
In [86]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)


         test_point_index = 1
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2595 0.0154 0.1339 0.4808 0.0215 0.0469 0.0073 0.0139 0.0208
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
```

#### 4.5.5.2. Incorrectly Classified point

```
In [87]: test_point_index = 100
         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respons
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         for i in indices:
             if i<9:
                 print("Gene is important feature")
             elif i<18:
                 print("Variation is important feature")
             else:
                 print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0166 0.2588 0.2318 0.0226 0.0307 0.0377 0.3495 0.0318 0.020
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
```

### 4.7 Stack the models

4.7.1 testing with hyper parameter tuning

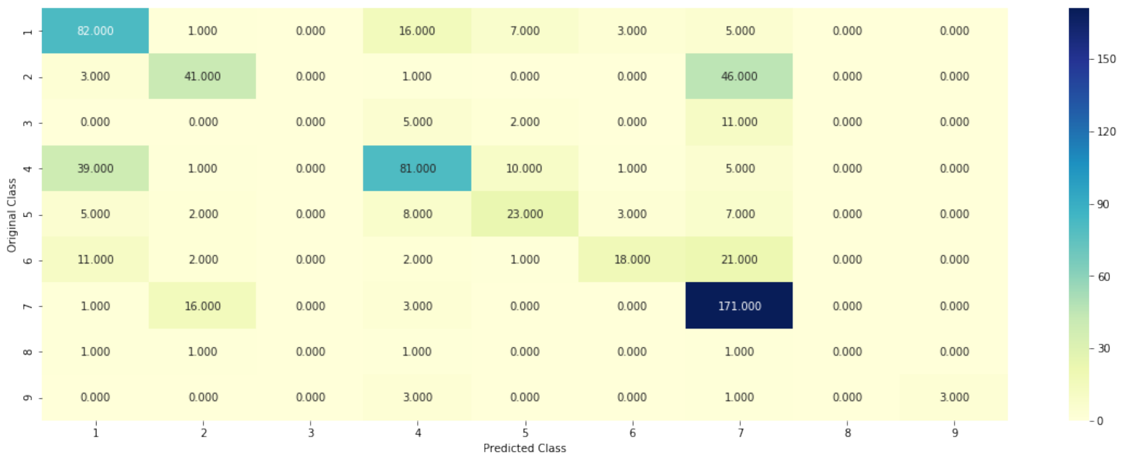
```
In [88]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, 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-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_sha

         # 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-lear
         # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=N
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
         # class_weight=None)

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

         # some of attributes of  RandomForestClassifier()
         # feature_importances_ : array of shape = [n_features]
         # The feature importances (the higher, the more important the feature).
         # --------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
         # --------------------------------


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

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


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

         sig_clf1.fit(train_x_onehotCoding, train_y)
         print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_prob
         sig_clf2.fit(train_x_onehotCoding, train_y)
         print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_p
         sig_clf3.fit(train_x_onehotCoding, train_y)
         print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_on
         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_classif
             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: 1.01
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.18
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.492
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.156
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.392
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.893
```

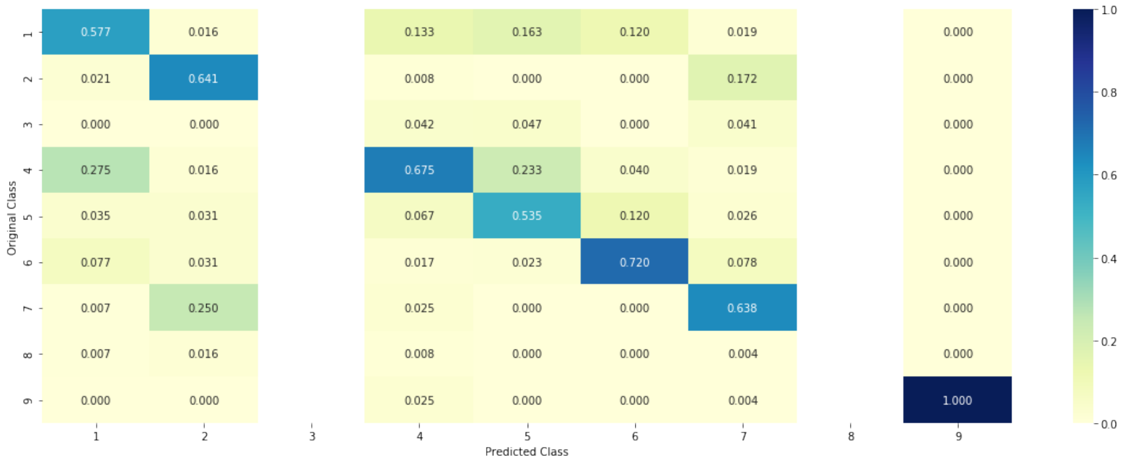
4.7.2 testing the model with the best hyper parameters

```
In [89]: 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 : 0.535551969273216
Log loss (CV) on the stacking classifier : 1.1562497109759189
Log loss (test) on the stacking classifier : 1.1719959433218796
Number of missclassified point : 0.3699248120300752
-------------------- Confusion matrix --------------------
```

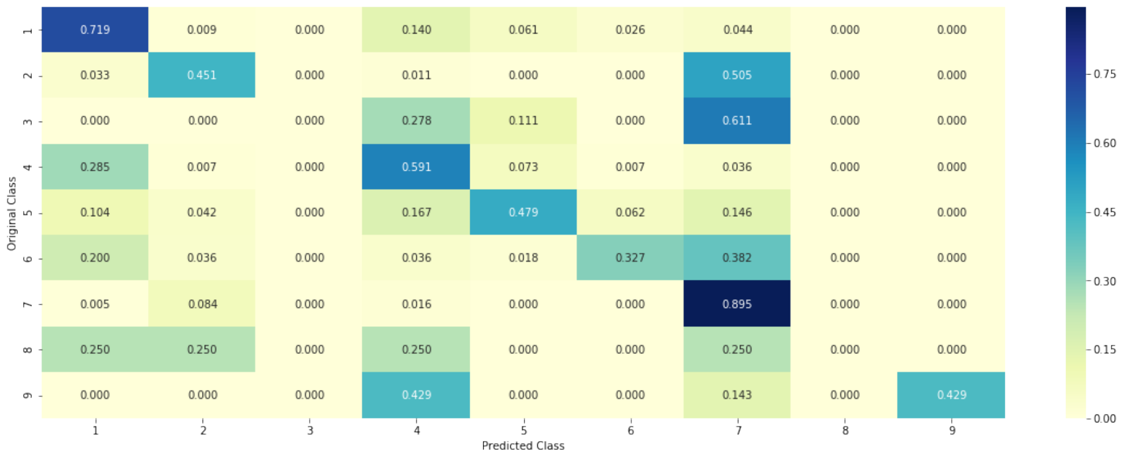


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.577 | 0.016 |  | 0.133 | 0.163 | 0.120 | 0.019 |  | 0.000 |
| 2 | 0.021 | 0.641 |  | 0.008 | 0.000 | 0.000 | 0.172 |  | 0.000 |
| 3 | 0.000 | 0.000 |  | 0.042 | 0.047 | 0.000 | 0.041 |  | 0.000 |
| 4 | 0.275 | 0.016 |  | 0.675 | 0.233 | 0.040 | 0.019 |  | 0.000 |
| 5 | 0.035 | 0.031 |  | 0.067 | 0.535 | 0.120 | 0.026 |  | 0.000 |
| 6 | 0.077 | 0.031 |  | 0.017 | 0.023 | 0.720 | 0.078 |  | 0.000 |
| 7 | 0.007 | 0.250 |  | 0.025 | 0.000 | 0.000 | 0.638 |  | 0.000 |
| 8 | 0.007 | 0.016 |  | 0.008 | 0.000 | 0.000 | 0.004 |  | 0.000 |
| 9 | 0.000 | 0.000 |  | 0.025 | 0.000 | 0.000 | 0.004 |  | 1.000 |

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

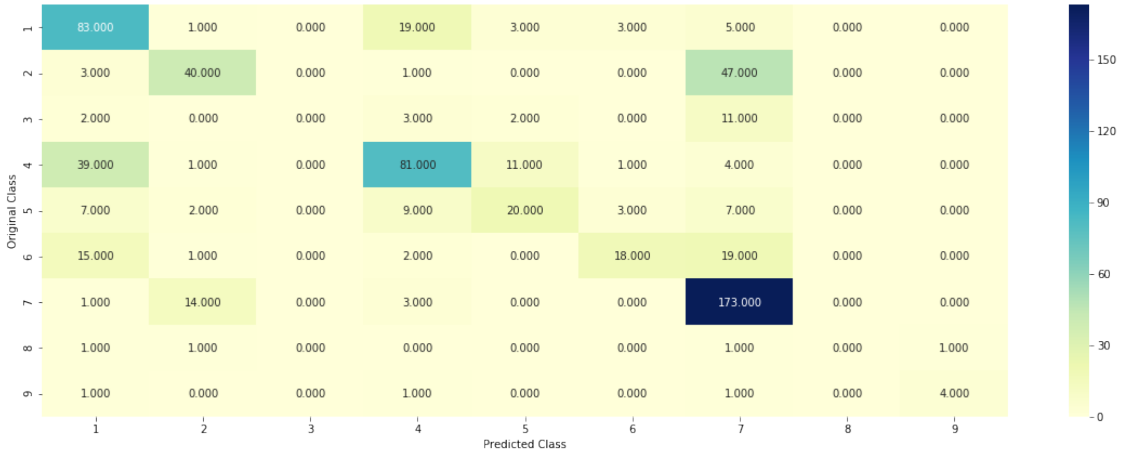


|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.719 | 0.009 | 0.000 | 0.140 | 0.061 | 0.026 | 0.044 | 0.000 | 0.000 |
| 2 | 0.033 | 0.451 | 0.000 | 0.011 | 0.000 | 0.000 | 0.505 | 0.000 | 0.000 |
| 3 | 0.000 | 0.000 | 0.000 | 0.278 | 0.111 | 0.000 | 0.611 | 0.000 | 0.000 |
| 4 | 0.285 | 0.007 | 0.000 | 0.591 | 0.073 | 0.007 | 0.036 | 0.000 | 0.000 |
| 5 | 0.104 | 0.042 | 0.000 | 0.167 | 0.479 | 0.062 | 0.146 | 0.000 | 0.000 |
| 6 | 0.200 | 0.036 | 0.000 | 0.036 | 0.018 | 0.327 | 0.382 | 0.000 | 0.000 |
| 7 | 0.005 | 0.084 | 0.000 | 0.016 | 0.000 | 0.000 | 0.895 | 0.000 | 0.000 |
| 8 | 0.250 | 0.250 | 0.000 | 0.250 | 0.000 | 0.000 | 0.250 | 0.000 | 0.000 |
| 9 | 0.000 | 0.000 | 0.000 | 0.429 | 0.000 | 0.000 | 0.143 | 0.000 | 0.429 |

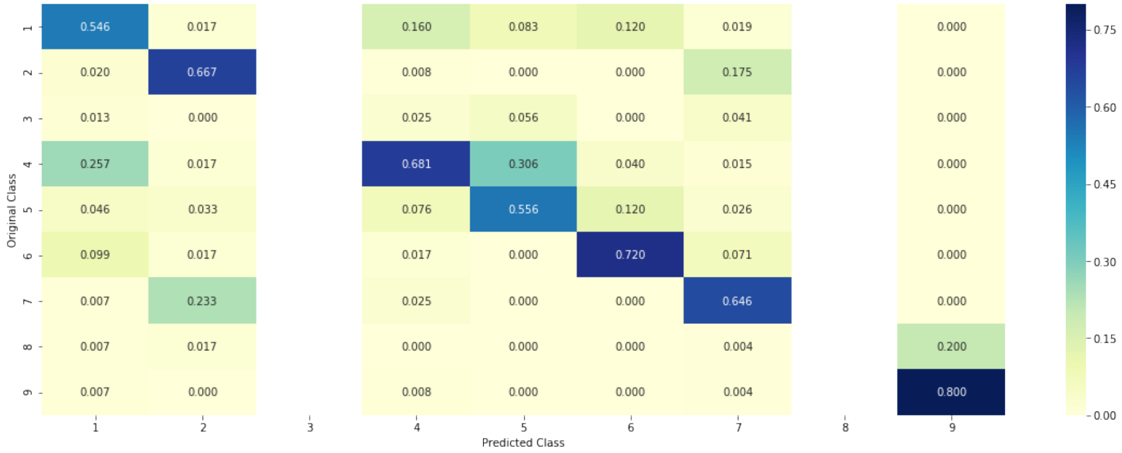
### 4.7.3 Maximum Voting classifier

In [90]:
```
#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_c
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_onehot
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

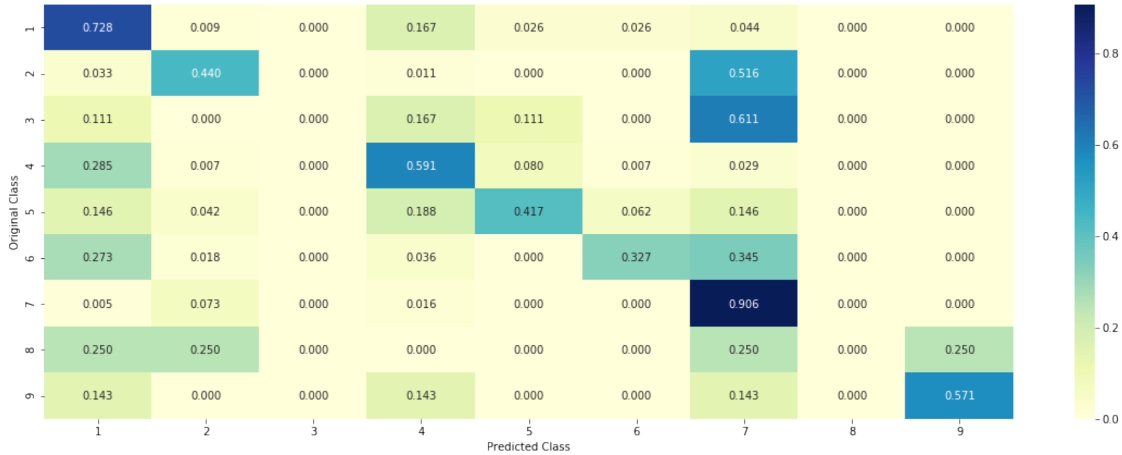
```
Log loss (train) on the VotingClassifier : 0.8311581097949535
Log loss (CV) on the VotingClassifier : 1.1739972098170757
Log loss (test) on the VotingClassifier : 1.182575428318276
Number of missclassified point : 0.3699248120300752
-------------------- Confusion matrix --------------------
```



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



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



## 0.1 Bigram Count Vectorizer with Logistic Regression

```
In [91]: # 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(train_df['TEXT'])
         # print(train_text_feature_onehotCoding.shape)
         # 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 : 803960
```

```
In [92]: train_text_features[:10], text_fea_dict.get('000')
```

```
Out[92]: (['00',
           '00 00',
           '00 0005',
           '00 006',
           '00 01',
           '00 02',
           '00 03',
           '00 05',
           '00 06',
           '00 08'],
          5836)
```

```
In [93]: dict_list = []
         # dict_list =[] contains 9 dictoinaries each corresponds to a class
         for i in range(1,10):
             cls_text = train_df[train_df['Class']==i]
             # build a word dict based on the words in that class
             dict_list.append(extract_dictionary_paddle(cls_text))
             # append it to dict_list

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


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

```
In [94]: len(confuse_array)

Out[94]: 803960
```

```
In [95]: # don't forget to normalize every feature
         train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

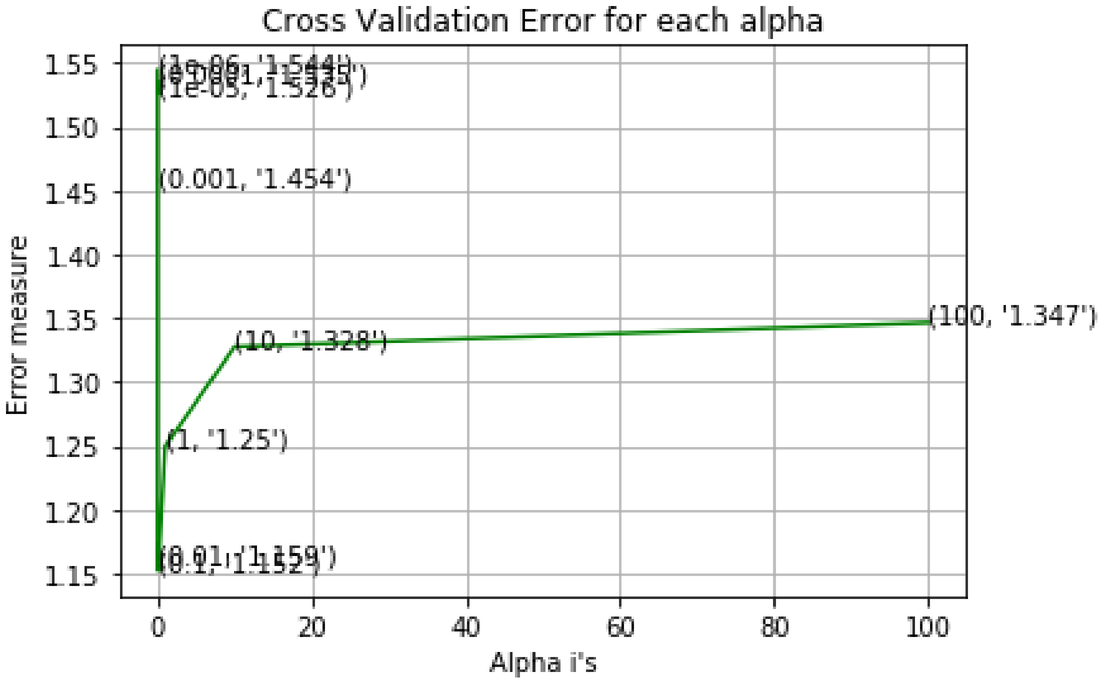
```
In [96]: # merging gene, variance and text features

         # 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_

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

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

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

### 0.2 Apply Logistic Regression

```
In [97]: alpha = [10 ** x for x in range(-6, 3)]
         print('alpha ',alpha)
         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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

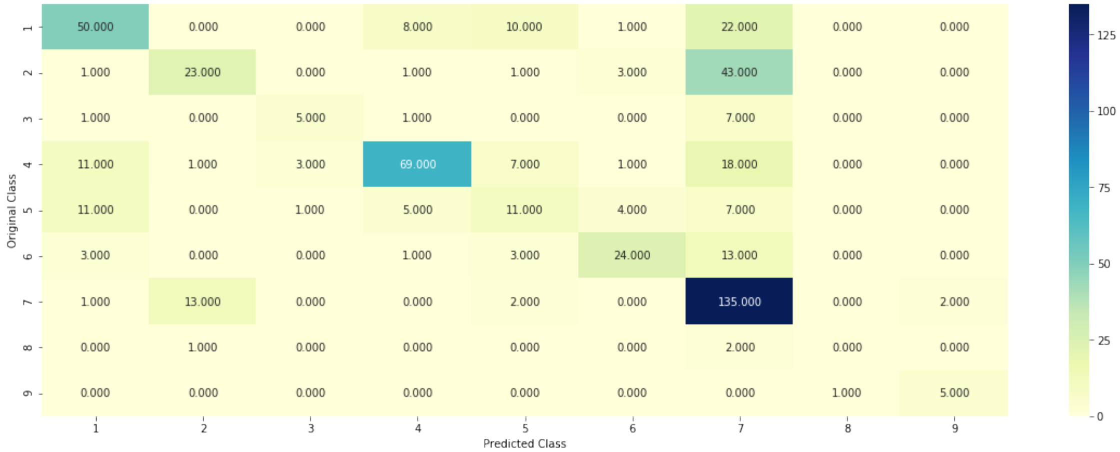
```
alpha  [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
for alpha = 1e-06
Log Loss : 1.5444196654896039
for alpha = 1e-05
Log Loss : 1.5264655718196316
for alpha = 0.0001
Log Loss : 1.5353204771419813
for alpha = 0.001
Log Loss : 1.453694031210995
for alpha = 0.01
Log Loss : 1.158894546919099
for alpha = 0.1
Log Loss : 1.1518830734110763
for alpha = 1
Log Loss : 1.2495390315995822
for alpha = 10
Log Loss : 1.3275090393110414
for alpha = 100
Log Loss : 1.3465194576247437
```



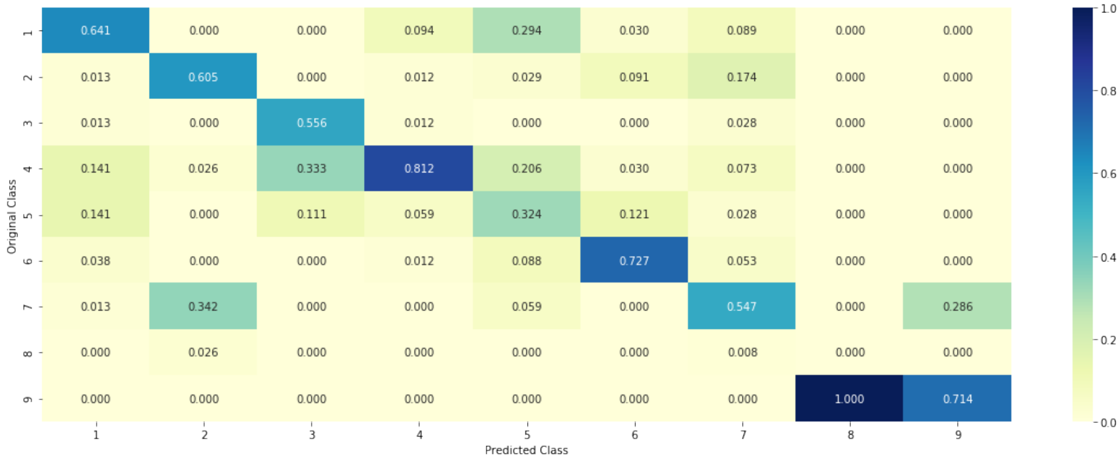
```
For values of best alpha =  0.1 The train log loss is: 0.856020140572819
For values of best alpha =  0.1 The cross validation log loss is: 1.1518830734110763
For values of best alpha =  0.1 The test log loss is: 1.1833367833247161
```

```
In [98]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
         predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv
```

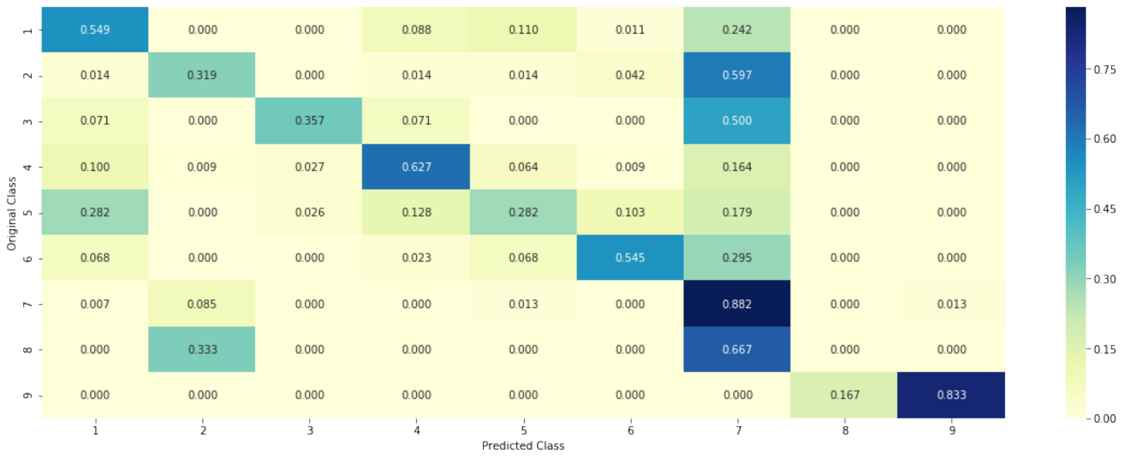
```
Log loss : 1.1518830734110763
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```



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



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



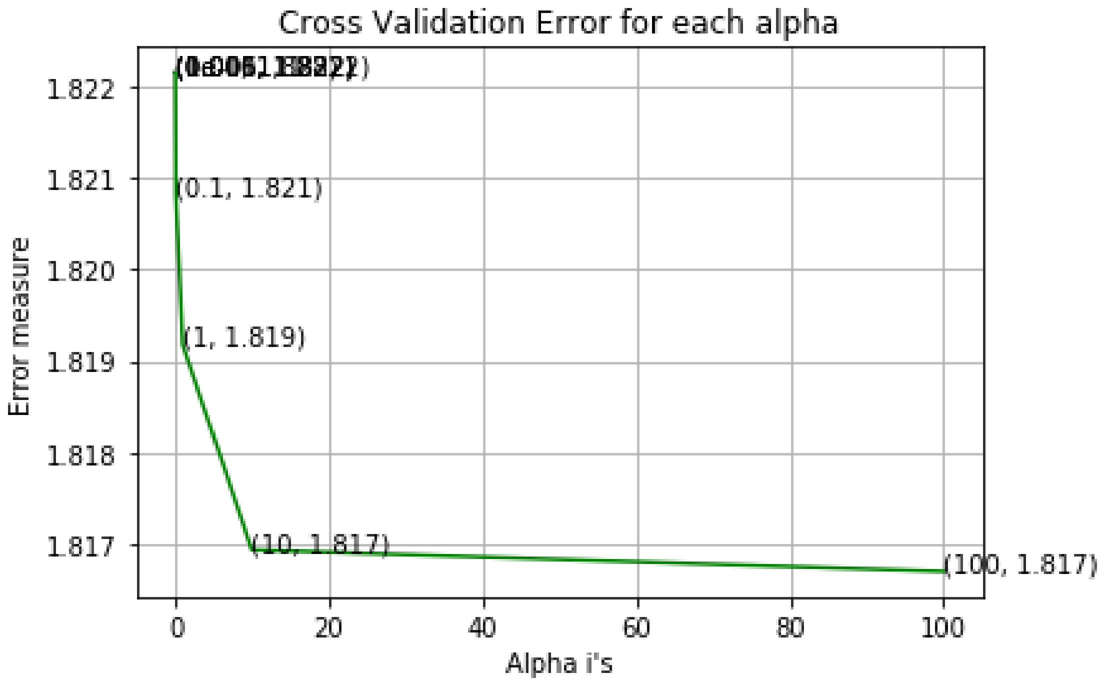
# 1 Feature Engineering

## 1.1 Extra Features

Trying 'number of word' and 'lenth of text' features with 'CountVectorizer'.

```
In [99]: temp_train_df = train_df.copy()
         temp_train_df['text_words'] = temp_train_df['TEXT'].apply(lambda row: len(row.split("
         temp_train_df['text_length'] = temp_train_df['TEXT'].str.len()


         temp_cv_df = cv_df.copy()
         temp_cv_df['text_words'] = temp_cv_df['TEXT'].apply(lambda row: len(row.split(" ")))
         temp_cv_df['text_length'] = temp_cv_df['TEXT'].str.len()

         temp_test_df = test_df.copy()
         temp_test_df['text_words'] = temp_test_df['TEXT'].apply(lambda row: len(row.split(" "
         temp_test_df['text_length'] = temp_test_df['TEXT'].str.len()

In [100]: temp_train_df = temp_train_df[temp_train_df['text_words'].notnull() ]
          temp_cv_df = temp_cv_df[temp_cv_df['text_words'].notnull()]
          temp_test_df = temp_test_df[temp_test_df['text_words'].notnull()]

          temp_train_df = temp_train_df[temp_train_df['text_length'].notnull() ]
          temp_cv_df = temp_cv_df[temp_cv_df['text_length'].notnull()]
          temp_test_df = temp_test_df[temp_test_df['text_length'].notnull()]
```

```
In [101]: from sklearn_pandas import DataFrameMapper

          mapper = DataFrameMapper([
               ('TEXT', CountVectorizer(ngram_range=(1,1), min_df=10)),
               ('text_words', None),
              ('text_length', None),

           ])

          train_features = mapper.fit_transform(temp_train_df)
          cv_features = mapper.transform(temp_cv_df)
          test_features = mapper.transform(temp_test_df)


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

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

          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=
          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_
          predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
          print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
          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_
For values of alpha =  1e-06 The log loss is: 1.822160579713747
For values of alpha =  1e-05 The log loss is: 1.8221605797145237
For values of alpha =  0.0001 The log loss is: 1.8221605797177962
For values of alpha =  0.001 The log loss is: 1.8221605797036327
For values of alpha =  0.01 The log loss is: 1.8221605797138882
For values of alpha =  0.1 The log loss is: 1.8208093577764521
For values of alpha =  1 The log loss is: 1.8191776924723901
For values of alpha =  10 The log loss is: 1.816933666180908
For values of alpha =  100 The log loss is: 1.8166951485444496
```



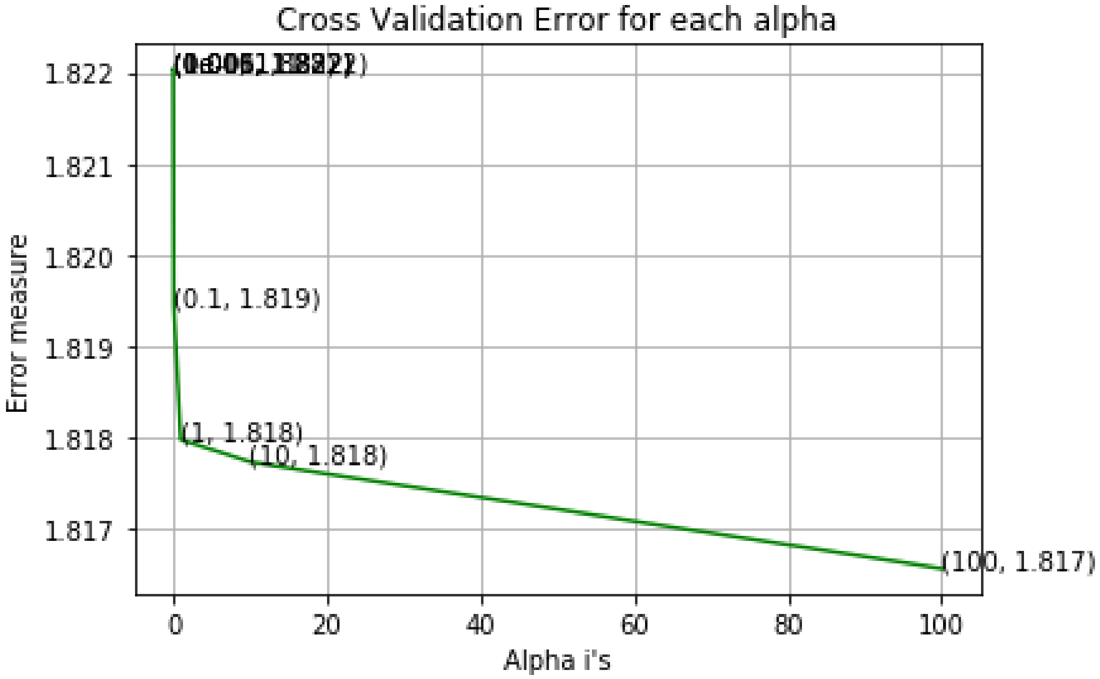
```
For values of best alpha =  100 The train log loss is: 1.2361499752558733
For values of best alpha =  100 The cross validation log loss is: 1.3699260961629411
For values of best alpha =  100 The test log loss is: 1.3815368312184526
```

Results are not quite good. Log loss is around 1.37 which bit higher.
**Let's try a tfidf vectorizer with 'no of words' and 'lenth of text' feature**

```
In [108]: mapper = DataFrameMapper([
               ('TEXT', TfidfVectorizer(ngram_range=(1,2), min_df=10)),
               ('text_words', None),
              ('text_length', None),

          ])
          train_features = mapper.fit_transform(temp_train_df)
          cv_features = mapper.transform(temp_cv_df)
          test_features = mapper.transform(temp_test_df)
```

```
In [109]: cv_log_error_array=[]
          for i in alpha:
              clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
              clf.fit(train_features, y_train)

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

          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=
          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
          predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
          print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
          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_
```

```
For values of alpha =  1e-06 The log loss is: 1.8220384848405864
For values of alpha =  1e-05 The log loss is: 1.822038484839043
For values of alpha =  0.0001 The log loss is: 1.822038484867094
For values of alpha =  0.001 The log loss is: 1.8220384848376174
For values of alpha =  0.01 The log loss is: 1.8220384848515465
For values of alpha =  0.1 The log loss is: 1.8194418728366117
For values of alpha =  1 The log loss is: 1.8179773184963033
For values of alpha =  10 The log loss is: 1.817733943204382
For values of alpha =  100 The log loss is: 1.816565409418684
```



```
For values of best alpha =  100 The train log loss is: 1.2361499752558733
For values of best alpha =  100 The cross validation log loss is: 1.3699260961629411
For values of best alpha =  100 The test log loss is: 1.3815368312184526
```

Here we can observe that 'no of words' and 'lenth of Text' are not good features. Count Vectorizer and TFIDF vectorizer both gives almost equal log loss so we can ignore those features.

A new idea is we can try **"FourGrams"**. But Using fourgram require high computational power, cause features will increase tremendous. So We will use **SelectKBest** to choose 1000 important feature along with TFIDF vectorizer.

```
In [91]: # building a CountVectorizer with all the words that occured minimum 3 times in train
         text_vectorizer = TfidfVectorizer(ngram_range=(1,4),min_df=10, max_features=1000)
         train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
         train_text_feature_onehotCoding = SelectKBest(chi2, k=1000).fit_transform(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*num
         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))

         # taking TOP 1000 features

         print("Total number of unique words in train data :", len(train_text_features))
         print('train_text_feature_onehotCoding shape ==>',train_text_feature_onehotCoding.shap
```

```
Total number of unique words in train data : 1000
train_text_feature_onehotCoding shape ==> (2124, 1000)
```

```
In [92]: train_text_features[:10], text_fea_dict.get('000')
```

```
Out[92]: (['000', '05', '10', '100', '11', '12', '13', '14', '15', '16'],
          14.129722873192303)
```

```
In [93]: dict_list = []
         # dict_list =[] contains 9 dictoinaries each corresponds to a class
         for i in range(1,10):
             cls_text = train_df[train_df['Class']==i]
             # build a word dict based on the words in that class
             dict_list.append(extract_dictionary_paddle(cls_text))
             # append it to dict_list

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


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

         # don't forget to normalize every feature
         train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

         # we use the same vectorizer that was trained on train data
```

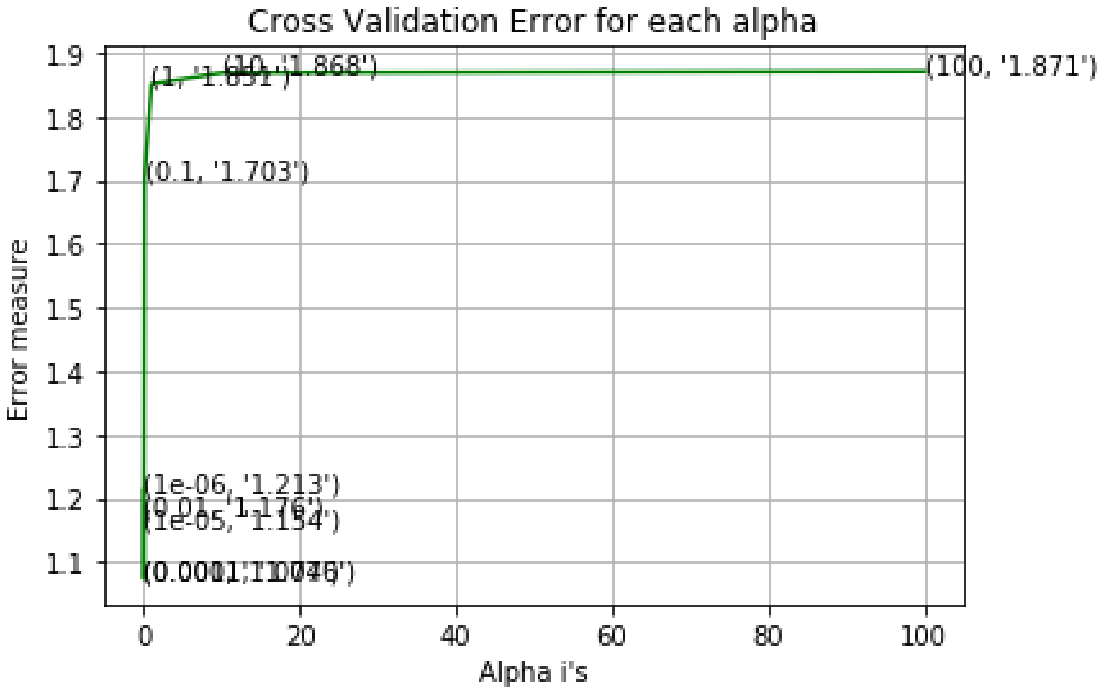
```
        cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
        # don't forget to normalize every feature
        cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

```
In [94]: 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_onehot
         train_y = np.array(list(train_df['Class']))

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

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

```
In [95]: alpha = [10 ** x for x in range(-6, 3)]
         print('alpha ',alpha)
         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_l
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
    predict_y = sig_clf.predict_proba(test_x_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

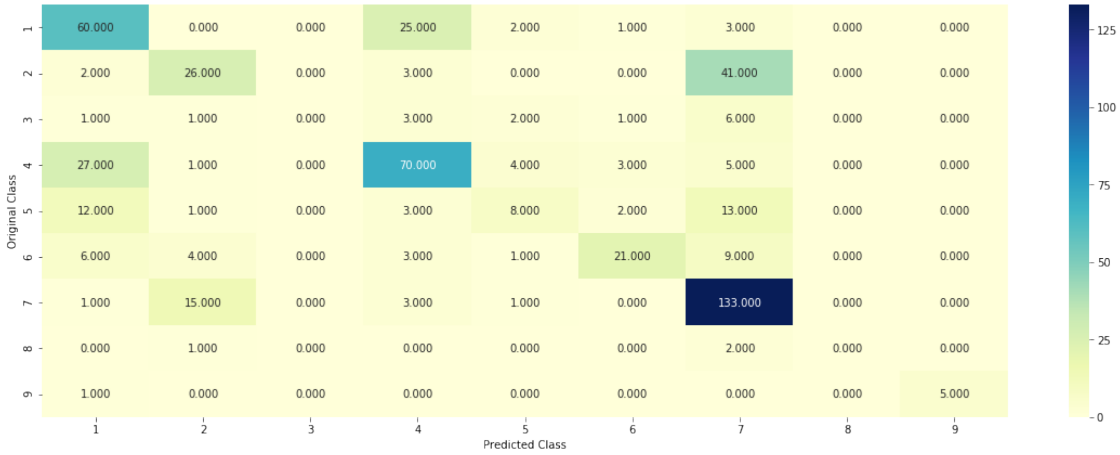
```
alpha  [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
for alpha = 1e-06
Log Loss : 1.212707277334664
for alpha = 1e-05
Log Loss : 1.153533111121014
for alpha = 0.0001
Log Loss : 1.0758569532173068
for alpha = 0.001
Log Loss : 1.073555694140625
for alpha = 0.01
Log Loss : 1.176356512596208
for alpha = 0.1
Log Loss : 1.7026559455757415
for alpha = 1
Log Loss : 1.8511449207433004
for alpha = 10
Log Loss : 1.8683802973850192
for alpha = 100
Log Loss : 1.8705021164147495
```



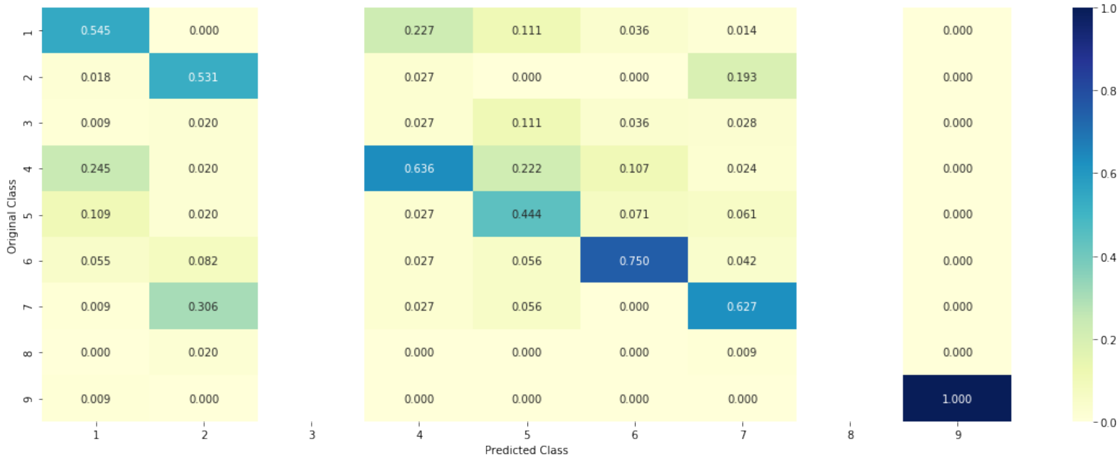
```
For values of best alpha =  0.001 The train log loss is: 0.7178898683114381
For values of best alpha =  0.001 The cross validation log loss is: 1.073555694140625
For values of best alpha =  0.001 The test log loss is: 1.025731799504609
```

```
In [96]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
         predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv
```

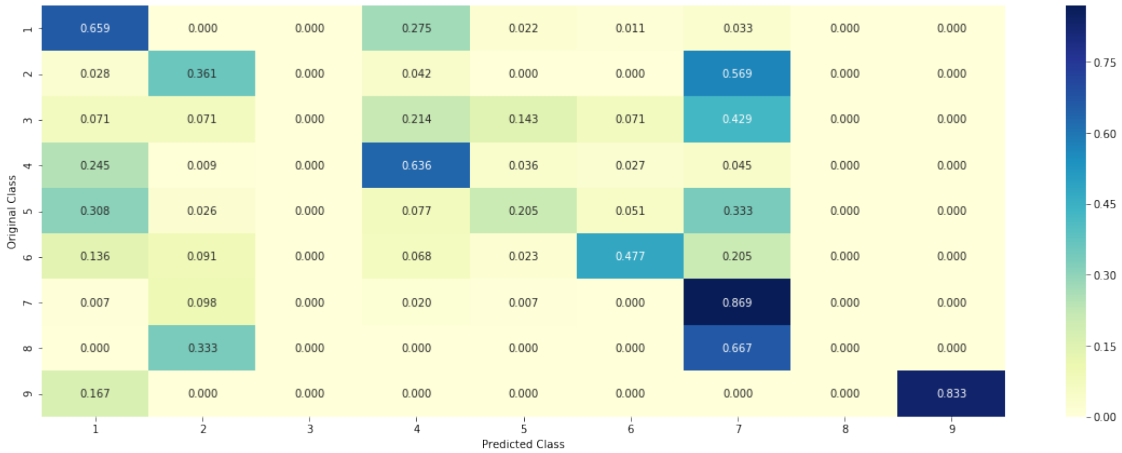
```
Log loss : 1.073555694140625
Number of mis-classified points : 0.39285714285714285
-------------------- Confusion matrix --------------------
```



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



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

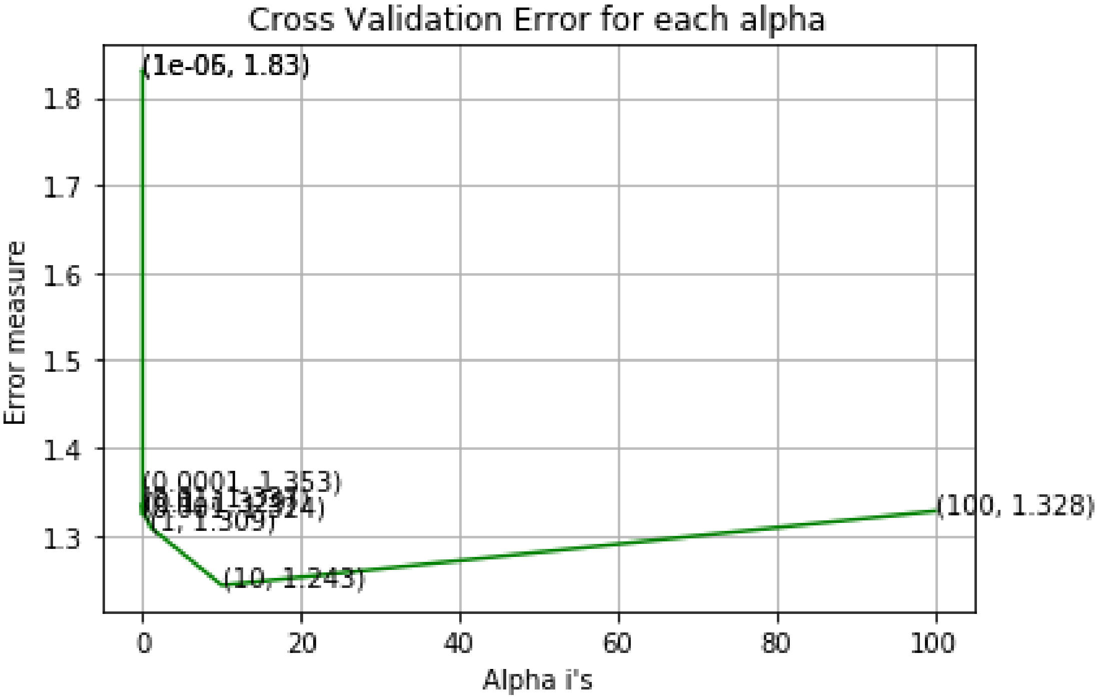


It can observe that log is quite low but yet it is not under 1. But it should notice that misclassified points percentage is incresing which is bad sign. RED ALERT.

We can try TDIDF and countvectorizer again it require higher computation power so we will limit ourselves to 1000 max features only.

```
In [100]: from sklearn_pandas import DataFrameMapper

          mapper = DataFrameMapper([
               ('TEXT', CountVectorizer(ngram_range=(1,4), min_df=10, max_features=1000)),
                  ('TEXT',TfidfVectorizer(ngram_range=(1,4),min_df=10, max_features=1000)),

           ])

          train_features = mapper.fit_transform(train_df)
          cv_features = mapper.transform(cv_df)
          test_features = mapper.transform(test_df)


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

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

          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=4
          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_
          predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
          print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
          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_l
```

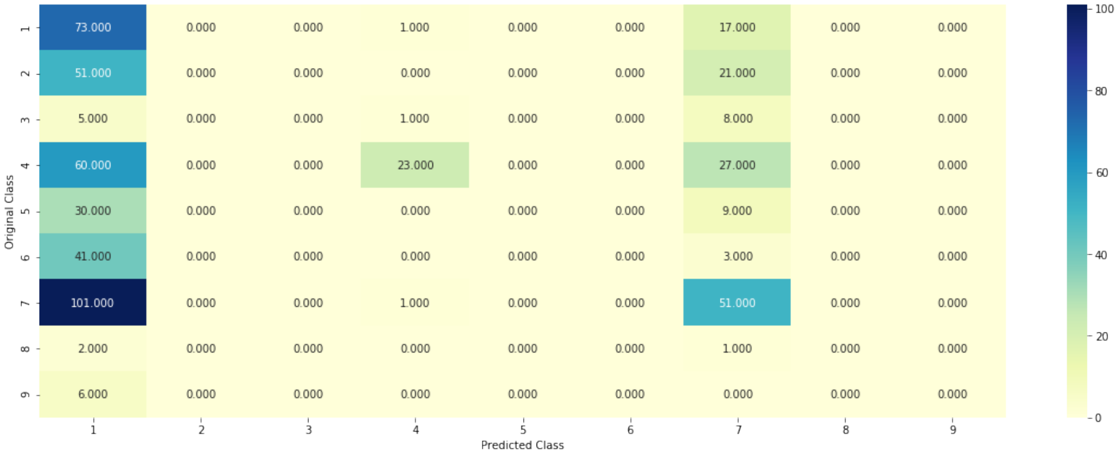
```
For values of alpha =  1e-06 The log loss is: 1.8304997567764278
For values of alpha =  1e-05 The log loss is: 1.8304997567764278
For values of alpha =  0.0001 The log loss is: 1.3526187637425466
For values of alpha =  0.001 The log loss is: 1.323785870393301
For values of alpha =  0.01 The log loss is: 1.3366673693797064
For values of alpha =  0.1 The log loss is: 1.3289461442950286
For values of alpha =  1 The log loss is: 1.309394611086164
For values of alpha =  10 The log loss is: 1.2431744140109304
For values of alpha =  100 The log loss is: 1.3277691185564866
```



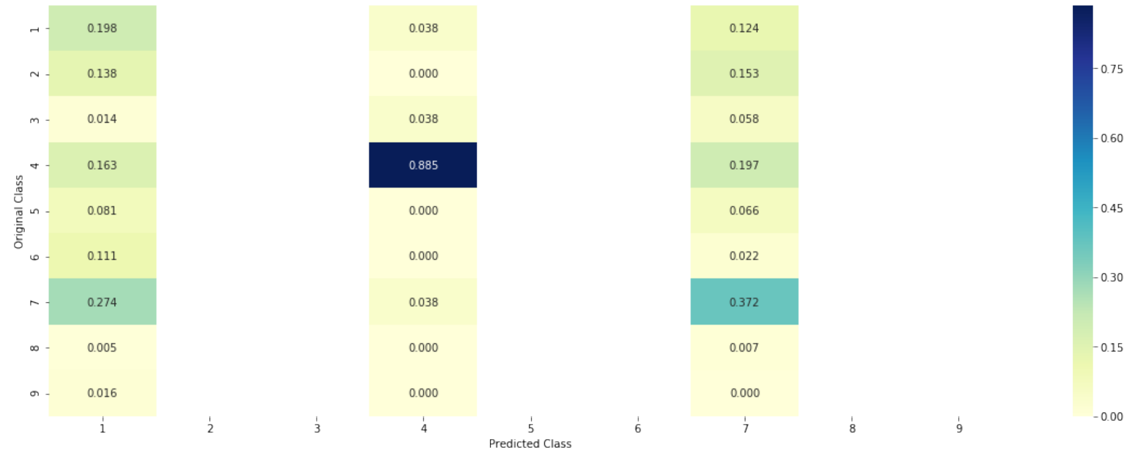
```
For values of best alpha =  10 The train log loss is: 1.78441584487254
For values of best alpha =  10 The cross validation log loss is: 2.1767942622361502
For values of best alpha =  10 The test log loss is: 1.9985137946444134
```

```
In [101]: 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
```

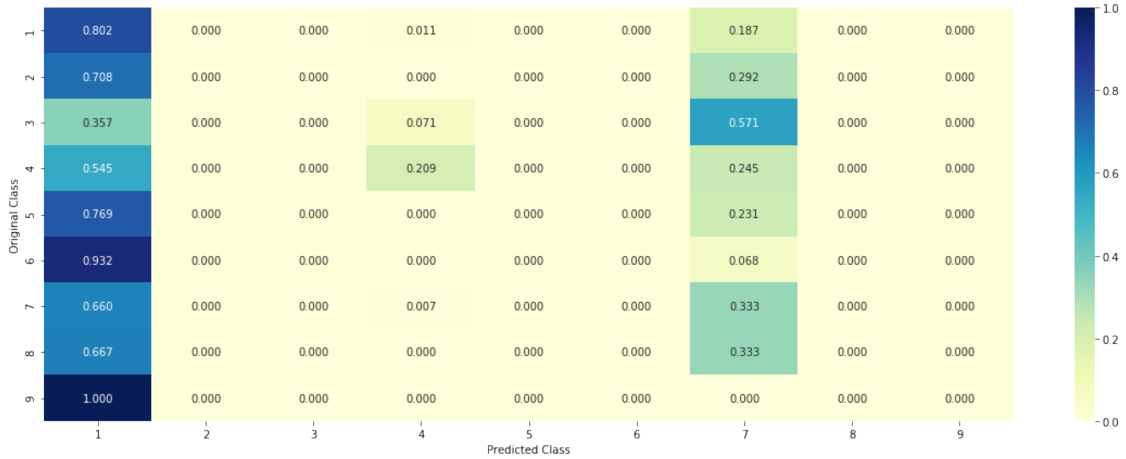
```
Log loss : 1.8683802973850192
Number of mis-classified points : 0.7236842105263158
-------------------- Confusion matrix --------------------
```



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



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



It was the worst model among all we tried. Probably because alone countvectorizer has 53k features and we are dealing with only 1k.

## 2 Conclusion

```
In [70]: import pandas as pd
         from prettytable import PrettyTable
         print('\t\t\t\t Model Comparision  ')
         print('\n')


         x = PrettyTable()
         x.field_names =  ['Metric','RM','NB','KNN', 'LR-bal', 'LR-w/o', 'LinearSVM', 'RF-res'

         x.add_row(["Tr Loss ",2.50,0.52342, 0.50799, 0.43916,0.42938, 0.47877,0.85274,0.05408
         x.add_row(["Test Loss ",2.50,1.18371, 1.00464,0.97336,1.01902,1.01222,1.17696,1.28650
         x.add_row(["Mis-class ",0.0,0.37781, 0.36278, 0.34774,0.34962,0.33646,0.40037,0.49624



         print('\n')
         print(x)
```

```
                                 Model Comparision



+------------+-----+---------+---------+---------+---------+-----------+---------+------------
|   Metric   |  RM |    NB   |   KNN   |  LR-bal |  LR-w/o | LinearSVM |  RF-res | RF-one hot
+------------+-----+---------+---------+---------+---------+-----------+---------+------------
|  Tr Loss   | 2.5 | 0.52342 | 0.50799 | 0.43916 | 0.42938 |  0.47877  | 0.85274 |  0.05408
| Test Loss  | 2.5 | 1.18371 | 1.00464 | 0.97336 | 1.01902 |  1.01222  | 1.17696 |   1.2865
| Mis-class  | 0.0 | 0.37781 | 0.36278 | 0.34774 | 0.34962 |  0.33646  | 0.40037 |  0.49624
+------------+-----+---------+---------+---------+---------+-----------+---------+------------
```

```
In [71]: x1 = PrettyTable()

         x1.field_names =  ['Metric','Vectorizer','Feature','TrLoss' ,'TestLoss','Mis-class']
         x1.add_row(["LR ",'Countvect','Bigram' ,0.85602, 1.15188, 0.39473])
         x1.add_row(["LR ",'Countvect','no_of words +text len' ,1.23614, 1.36992, 0.38873])
         x1.add_row(["LR ",'TFIDF','no_of words +text len' ,1.23614, 1.36992, 0.38873])
         x1.add_row(["LR ",'TFIDF','fourGrams' ,0.71788, 1.07355, 0.3928])
         x1.add_row(["LR ",'TFIDF + CountVect','--' ,1.78441, 2.176795, 0.72368])

         print(x1)
```

```
+--------+-------------------+-----------------------+---------+----------+-----------+
| Metric |     Vectorizer    |        Feature        |  TrLoss | TestLoss | Mis-class |
+--------+-------------------+-----------------------+---------+----------+-----------+
|  LR    |     Countvect     |         Bigram        | 0.85602 | 1.15188  |  0.39473  |
|  LR    |     Countvect     | no_of words +text len | 1.23614 | 1.36992  |  0.38873  |
|  LR    |       TFIDF       | no_of words +text len | 1.23614 | 1.36992  |  0.38873  |
|   LR   |       TFIDF       |       fourGrams       | 0.71788 | 1.07355  |   0.3928  |
|   LR   | TFIDF + CountVect |          --           | 1.78441 | 2.176795 |  0.72368  |
+--------+-------------------+-----------------------+---------+----------+-----------+
```

- We can say that Logistic regression model wtih **SelectBestK** is 1000 is far better model which test log loss under 1.000.
- We tried feature engineering, TFIDF with fourgram seems good, but due to computational power consumption we limit ourselve to some set of points we can perform the same with better computational power or more optimized way.